Analysis of a Nuclear Reactor Boilure Closure Unit Through Development of a 3D Parallel Finite Element Code

van Griethuysen, Lorna Charlotte

ANALYSIS OF A NUCLEAR REACTOR BOILER CLOSURE UNIT THROUGH DEVELOPMENT OF A 3D PARALLEL FINITE ELEMENT CODE

Lorna C. van Griethuysen

May 15, 2012

Summary

Three dimensional (3D) finite element analysis (FEA) allows the mechanical integrity of complex structures to be estimated with some confidence. This research is concerned with extending an existing parallel FEA code. This code has been run on up to 16 processors on Durham University's high performance computing (HPC) cluster and two different parallel linear solvers have been compared. A key feature of the work has been to develop tools for structural analyses. An automatic mesh refinement program has been written, the Zienkiewicz and Zhu error estimator has been coded for 3D hexahedral meshes and post processing techniques have been used to calculate and visualise principal stress data and peak stress criteria. This project also reports on three peak stress envelopes used to assess the condition of a concrete sub-structure.

The development of this parallel code has enabled the deformation behaviour of a key component of a nuclear rector vessel to be determined. The BCU is a prestressed cylindrical concrete vessel (depth of $1.73m$ and diameter of $3.37m$) sealing the top of a boilers housed within the walls of the reactor. In recent years possible problems have been identified at the Hartlepool and Heysham I Advance Gas-Cooled nuclear reactors (AGR) with respect to the structural condition of the BCU (in particular the condition of the prestressed circumferential wires designed to maintain the BCU in a state of compression). This problem provides an interesting case study for this project. Four different BCU meshes have been used containing either 40201 or 321608 elements (the elements are either 8 or 20-noded hexahedral elements). Three different load cases have been used to model the BCU. The results of the analyses confirm that the circumferential pre-stressing is vital in order to keep the BCU is a state of compression and operating under safe working conditions. These results have been confirmed using principal stress plots and three different peak stress envelopes. The results show that when the pre-stressing is released approximately one quarter of the elements contain stresses at Gauss points which exceed the peak strength of the concrete. This suggests that under these extreme conditions the BCU's structural integrity has been severely compromised, concrete rupture is possible and the nuclear reactor is no-longer safe to operate.

Acknowledgements

The author would like to thank the following people for their support and encouragement throughout the project:

Prof. Roger Crouch for his guidance and stimulation throughout this project, and for all the time given over to regular meetings. Also Dr. Ashraf Osman who provided some supervision.

Henk Slim for his direction on how to use Durham University's High Performance Computing Service and his willingness to sort out computer problems.

Lastly the author would like to thank other member of the Computational Mechanics Group in Durham for their support in the office. In particular, to William Coombs and Zahur Ullah for their willingness to talk over problems, share knowledge and give advice.

Contents

Summary i

Acknowledgments ii

List of Figures vi

List of Tables x

Nomenclature & abbreviations xi

1 Introduction 1

1.1 Scope of the thesis and background to the study 1

1.2 Project aims 2

1.3 Thesis breakdown 2

2 Finite element analysis 4

2.1 Equilibrium 4

2.2 Constitutive relationships 6

2.3 Strain-displacement relationships 6

2.4 Finite element equations 7

2.5 Isoparametric hexahedral elments 8

2.5.1 Local co-ordinate system and the co-ordinate mapping technique 8

2.5.2 Shape functions and their derivatives 10

2.6 Non-linear finite element analysis 11

3 Parallel finite element analysis 14

3.1 Parallel computing 14

3.1.1 Message Passing Interface (MPI) 16

3.1.2 The high performance computing service in Durham 18

3.2 Parallel finite element analysis . . . 19
3.2.1 ParaFEM . . . 20
3.2.2 Linear solvers . . . 21
3.2.3 Pre-conditioned conjugate gradient solver (PCCG) . . . 21
3.2.4 MUMPS . . . 22

4 Code development: extra features added to ParaFEM 26
4.1 Self weight . . . 27
4.1.1 Self weight example . . . 28
4.2 Pressure loading . . . 28
4.2.1 Pressure loading example . . . 30
4.3 Mesh refinement . . . 31
4.3.1 p-refinement: 8 $\rightarrow$ 20-noded elements . . . 33
4.3.2 h-refinement: each 20-noded element $\rightarrow$ eight 8-noded elements . . . 34
4.3.3 Local mesh refinement . . . 35
4.4 Error analysis . . . 38
4.4.1 Calculating recovered nodal stresses within a 3D mesh using SPR . . . 39
4.4.2 SPR error analysis example . . . 43
4.5 Post processing . . . 46
4.5.1 Peak stress envelopes . . . 46
4.5.2 A smooth peak stress envelope for structural concrete . . . 49
4.5.3 Comparing TCenv, MPenv and Senv . . . 50
4.5.4 Introduction to Gmsh . . . 51
4.5.5 Principal stress plots . . . 52

5 Nuclear reactor BCU: specific application for the developed ParaFEM code 53
5.1 The boiler closure unit . . . 53
5.2 BCU mesh generation . . . 56
5.3 BCU loading conditions . . . 56
5.3.1 Underside pressure and associated loading within each penetration . . . 58
5.3.2 Bolt loads . . . 61
5.3.3 Pre-stress due to radial wire windings . . . 61
5.3.4 Weight of boiler . . . 62
5.3.5 Self weight . . . 62
5.4 Using different linear solvers for BCU FEAs . . . 63

5.4.1 BCU FEAs run time comparisons . . . 64
5.5 Running a BCU parallel FEA . . . 64
5.5.1 Steps required to further refine the BCU mesh . . . 64
5.6 Results . . . 66
5.6.1 Displacements . . . 66
5.6.2 Peak stress envelopes . . . 71
5.6.3 How non-linear FEA could alter the findings on the structural condition of the BCU 73
5.6.4 Principal stress plots . . . 81
5.6.5 Error analysis . . . 86

6 Conclusions and recommendations for further work 88

A `ParaFEM` Input Files 90
A.1 `p.121.dat` . . . 91
A.2 `p.121.d` . . . 92
A.3 `p.121.bnd` . . . 94
A.4 `p.121.lds` . . . 95
A.5 `p.121.prs` . . . 96

B `make` File Example 97

C Code Appendix 100
C.1 ParaFEM - with MUMPS solver . . . 100

List of Figures

2.1 Stresses at a point in equilibrium - considering the x direction. 5

2.2 Mapping from the local to the global domain 9

2.3 Node numbering for 8 and 20-noded element shape functions 10

2.4 Idealised stress-strain curves for concrete under one-dimensional loading 12

3.1 Shared memory model 15

3.2 Message passing model 16

4.1 Boundary conditions for the unit cube used in the example problems 26

4.2 Self weight capability example problems 28

4.3 Integration schemes for pressure loading 29

4.4 Uniform pressure load on the example cube 31

4.5 Gauss point principal stress vectors resulting from a uniform pressure load 31

4.6 Linearly varying pressure acting on the example cube 32

4.7 Gauss point stresses due to a linearly varying pressure load 32

4.8 Mesh refining process 33

4.9 `etpl` node ordering 33

4.10 New nodes for 8 x 8-noded mesh 34

4.11 Sub-elements 1-8 node numbering scheme 35

4.12 Unconforming hexahedral 8-noded mesh 36

4.13 3-Ref example 37

4.14 3-Ref unstable template 37

4.15 *Pillowing* of a quadrilateral mesh [7] 38

4.16 Four 2-Ref refinement templates by Schneiders *et al* for (a) node; (b) edge; (c) face; (d) volume 38

4.17 Three 3-Ref refinement templates by Schneiders *et al* for (a) A; (b) B; (c) C and (d) D . . 38

4.18 Setting up 8 and 20-noded patches within the domain 40

4.19 2D quadrilateral element patch extensions at domain boundaries 41
4.20 3D hexahedral element patch extensions at domain boundaries 41
4.21 SPR Gauss point integration schemes for hexahedral patches 43
4.22 Infinite plate with a circular hole example problem 43
4.23 In-plane stresses at point X for meshes 1 to 5 44
4.24 Energy norm error, $|e|$, visualisation for Mesh 3 45
4.25 Bi-axial peak stress envelope showing tensile and compressive behaviour [5] 46
4.26 Concrete peak stress envelope 48
4.27 Figure to show how to determine whether stress state is inside or outside the peak stress envelope 49
4.28 Meridional section for Senv 50
4.29 Meridional section for Senv, MPend, TCenv 51

5.1 Hartlepool/Heysham I PCPV boiler details [1] 54
5.2 Hartlepool/Heysham I PCPV BCU: Sectional elevation. The concrete is identified by the darker grey shading. [1] 55
5.3 Photograph of pre-stressed wires being wound around the BCU [1] 55
5.4 BCU mesh showing loading conditions and co-ordinate system used 57
5.5 Deformed mesh with nodal ring loading due to maximum underside pressure (deformation scale factor 200) 59
5.6 Deformed mesh with linearly varying shear forces acting inside the penetrations (deformation scale factor 200) 60
5.7 Forces acting on the BCU due to the pre-stressed wire windings 61
5.8 Identification of three BCU nodes and radial and tangential displacement directions . . . 67
5.9 Exaggerated deformation plots: view 1 (deformation scale factor 400) 70
5.10 Exaggerated deformation plots: view 2 (deformation scale factor 400) 70
5.11 Exaggerated deformation plots: view 3 (deformation scale factor 400) 70
5.12 Deformation mesh plots (scale factor 200) showing the elements where the stresses lie outside the peak stress envelopes: mesh 8, load case 1 74
5.13 Deformation mesh plots (scale factor 200) showing the elements where the stresses lie outside the peak stress envelopes: mesh 8, load case 2 74
5.14 Deformation mesh plots (scale factor 200) showing the elements where the stresses lie outside the peak stress envelopes: mesh 8, load case 3 74
5.15 Deformation mesh plots (scale factor 200) showing the elements where the stresses lie outside the peak stress envelopes: mesh 20, load case 1 75
5.16 Deformation mesh plots (scale factor 200) showing the elements where the stresses lie outside the peak stress envelopes: mesh 20, load case 2 75
5.17 Deformation mesh plots (scale factor 200) showing the elements where the stresses lie outside the peak stress envelopes: mesh 20, load case 3 75

5.18 Deformation mesh plots (scale factor 200) showing the elements where the stresses lie outside the peak stress envelopes: mesh 88, load case 1 76

5.19 Deformation mesh plots (scale factor 200) showing the elements where the stresses lie outside the peak stress envelopes: mesh 88, load case 2 76

5.20 Deformation mesh plots (scale factor 200) showing the elements where the stresses lie outside the peak stress envelopes: mesh 88, load case 3 76

5.21 Deformation mesh plots (scale factor 200) showing the elements where the stresses lie outside the peak stress envelopes: mesh 820, load case 1 77

5.22 Deformation mesh plots (scale factor 200) showing the elements where the stresses lie outside the peak stress envelopes: mesh 820, load case 2 77

5.23 Deformation mesh plots (scale factor 200) showing the elements where the stresses lie outside the peak stress envelopes: mesh 820, load case 3 77

5.24 Deformation mesh plots (scale factor 200) showing the elements where the stresses lie outside the peak stress envelopes: mesh 8, load cases 1-3 (see the colour code given on Figure 5.28) 78

5.25 Deformation mesh plots (scale factor 200) showing the elements where the stresses lie outside the peak stress envelopes: mesh 20, load cases 1-3 (see the colour code given on Figure 5.28) 78

5.26 Deformation mesh plots (scale factor 200) showing the elements where the stresses lie outside the peak stress envelopes: mesh 88, load cases 1-3 (see the colour code given on Figure 5.28) 78

5.27 Deformation mesh plots (scale factor 200) showing the elements where the stresses lie outside the peak stress envelopes: mesh 820, load cases 1-3 (see the colour code given on Figure 5.28) 79

5.28 Peak stress envelope indicators used to identify where the stresses exceed the strength limit 79

5.29 Meridional section for Senv showing location of stress states which lie outside MPenv . . . 80

5.30 Meridional section for Senv showing location of stress states which lie outside MPenv and inside Senv 80

5.31 Meridional section for Senv showing location of stress states which lie inside MPenv and outside Senv 80

5.32 Tensile principal stress vectors located outside MPenv - load case 1 82

5.33 Tensile principal stress vectors located outside MPenv - load case 2 83

5.34 Tensile principal stress vectors located outside MPenv - load case 3 84

5.35 Cross section A: tensile and compressive principal stress vectors near the top surface of BCU (at depth $y = 1697$) 85

5.36 Cross section B: tensile and compressive principal stress vectors in the middle of the BCU (at depth $y = 866$) 85

5.37 Cross section C: tensile and compressive principal stress vectors near the bottom of the BCU (at depth $y = 111$) 85

5.38 Location of the element where the maximum error in energy norm occurs for each BCU mesh 87

A.1 Example problem 90

A.2 Example `p121.dat` file 91
A.3 Example `p121.d` file 93
A.4 Example `p121.bnd` file 94
A.5 Example `p121.lds` file 95
A.6 Example `p121.prs` file 96

B.1 Example `make` file (a) 98
B.2 Example `make` file continued (b) 99

List of Tables

3.1 `ParaFEM` input file summary 20

3.2 List of `MUMPS` variables for distributed assembled problems 24

4.1 Self weight example problem results (total forces) 28

4.2 Table to show the verteces associated with each new edge node for p-refinement 34

4.3 Table to show the vertex nodes associated with each new surface node for h-refinement . . 34

4.4 SPR example 8-noded element results 45

4.5 SPR example 20-noded element results 45

4.6 TCenv constants 47

4.7 Additional constants for MPenv 47

5.1 No. of elements and nodes in each BCU mesh 56

5.2 BCU pre-stress wire loading contributions 62

5.3 Summary of the number of nodal loads and pressurised faces for each BCU load case and mesh 63

5.4 Linear solver run time comparisons for BCU FEA (load case 2) 65

5.5 Displacement recorded in mm for node 795 of the BCU meshes 67

5.6 Displacement recorded in mm for node 560 of the BCU meshes 68

5.7 Displacement recorded in mm for node 6956 of the BCU meshes 69

5.8 Table to show the number of elements with principal stresses lying outside the peak stress envelopes 71

5.9 BCU SPR error analysis results 86

A.1 Variables listed in `p121.dat` 91

A.2 Variable listed in `p121.d` 92

A.3 Variables listed in `p121.bnd` 94

A.4 Variables listed in `p121.lds` 95

A.5 Variables listed in `p121.prs` 96

Nomenclature and abbreviations

Firstly non-Greek symbols will be listed in alphabetical order. Scalars, vectors and tensors will be grouped separately. Greek symbols will then be listed in the same way followed by mathematical operators and abbreviations.

d_r	Radial displacement
d_t	Tangential displacement
u	Displacement in the global direction x
v	Displacement in the global direction y
w	Displacement in the global direction z
c	Circumference
d	Total length of BCU penetration
f	Yield function
$\overline{f}_{bc}$	Equal biaxial compression, normalised with respect to f_c
f_c	Uniaxial compressive strength
f_{ext}	External force
$\overline{f}_{ht}$	Equal hydrostatic tension, normalised with respect to f_c
$\overline{f}_t$	Uniaxial tensile strength, normalised with respect to f_c
f_t	Uniaxial tensile strength
g	Acceleration due to gravity
h	Height of BCU
l	Distance to a point on a plate from the origin
p	Pressure
$\hat{r}$	Deviatoric shape function
r	Radius
r_0	Smooth peak stress envelope (Senv) variable used to calculate $\hat{r}$
r_1	Senv variable variable used to calculate $\hat{r}$
r_2	Senv variable variable used to calculate $\hat{r}$
t	Thickness of radial wire windings
x	Global co-ordinate direction
y	Global co-ordinate direction
z	Global co-ordinate direction
A	Area
E	Young's modulus
F_b	Force due to bolt loads
F_i	Force due to shear stresses in BCU inlets
F_{sw}	Force due to self weight of the BCU
F_u	Force on BCU due to underside pressure
F_c	Force due to wire windings
G	Gravitational force
P_z	Nodal equivalent force in z direction due to self weight
V	Volume
$\{a\}$	Polynomial coefficients vector
$\{d\}$	Displacement
$\{e_\sigma\}$	Error in stress
$\{f_b\}$	Body force
$\{\hat{n}\}$	Unit vector normal to a plane
$\{n\}$	Vector normal to a plane
$\{\hat{p}\}$	Polynomial terms
$[B]$	Strain-displacement matrix
$[D]$	Constitutive matrix
$\{d^e\}$	Element displacement vector
$\{f^e\}$	Element force vector
$[I]$	Identity matrix
$[J]$	Jacobian matrix
$[K]$	Global stiffness matrix
$[K^e]$	Element stiffness matrix
$[L]$	Matrix of differential operators
$[N]$	Shape function matrix
$\{P\}$	PCCG vector
$\{Q\}$	PCCG vector
$\{R\}$	PCCG residual vector
$\{U\}$	PCCG solution vector
D_{ijkl}	Constitutive tensor
ρ	Stress invariant
α	Senv variable
$\hat{\alpha}$	PCCG scalar
$\hat{\beta}$	PCCG scalar
ν	Poisson's ratio
ω	Integration weights

ω^*	Virtual displacement
ρ	Density of concrete
ϱ	Stress invariant
ϱ_c	Compression meridian
ϱ_e	Extension meridian
σ_c	Stress in pre-stress wires
φ	Lode angle
$\hat{\xi}$	Local co-ordinate direction
ξ	Stress invariant
$\hat{\eta}$	Local co-ordinate direction
$\hat{\zeta}$	Local co-ordinate direction
Ω	Domain of a body in the reference configuration
Θ	Angle
$\{\epsilon\}$	Strain vector
$\{\sigma\}$	Stress vector
$\{\sigma^*\}$	Exact stress vector
$\{\sigma_h\}$	FEA stress vector
ϵ_{kl}	Stress tensor
σ_{ij}	Stress tensor
$\frac{\delta}{\delta a}(\cdot)$	Derivative of $(\cdot)$ with respect to a
$\det(\cdot)$	Determinant of $(\cdot)$
$\triangle(\cdot)$	Increment of $(\cdot)$
$(\cdot)^T$	Transpose of $(\cdot)$

AGR	Advanced gas cooled nuclear reactor
BCU	Boiler closure unit
CIS	Computer Information Service
FEA	Finite element analysis
FE	Finite element
FEM	Finite element method
GUTS	Guaranteed ultimate tensile strength
HPS	High performance computing
MPenv	Multi-planar peak stress envelope
MPI	Message passing interface
nGp	Number of Gauss points per element
$nGppP$	Number of Gauss points per SPR patch
PCCG	Pre-conditioned conjugate gradient
PCPV	Pre-stressed concrete pressure vessel
PFEA	Parallel finite element analysis
Senv	Smooth C2 peak stress surface
SPR	Superconvergent patch recovery
ssh	Secure shell
TCenv	Tension cut-off peak stress envelope
Z^2	Zienkiewicz Zhu

Chapter 1

Introduction

1.1 Scope of the thesis and background to the study

The pre-stressed boilure closure unit (BCU) provides a pressure barrier to the boiler within an advanced gas cooled nuclear reactor (AGR). The BCUs are subject to high gas pressure. Good structural condition of these units is vital for the safety of the reactor. This structure provides an application for the code development featured in this project. Finite element analysis (FEA) studies were initially carried out on the BCU around four years ago when four AGR reactors had to be shut down (Hartlepool and Heysham I). These shut-downs were due to concern over the structural condition of the BCU, in particular what would happen if the pre-stressed wire windings snapped or their condition deteriorated. The shut down was initiated when a broken wire was suspected on a BCU at Hartlepool I during a planned inspection in September 2007. This project has further investigated this problem, looking at how the code could (i) be run faster using different solvers and (ii) more accurately using different meshes, while obtaining an error estimation for each mesh and evaluating the structural condition using different post processing procedures.

FEA is widely used in industry for stress and deformation analysis, allowing the structural integrity and safety of the BCU to be evaluated. Speed and accuracy are two factors which must be balanced when carrying out FEA within a fixed time frame and budget. Most of the run time in a large FEA is expended on solving the system of linear equations. This increasingly becomes the case as the mesh is refined, therefore it is advantageous to implement parallel solvers. Greater accuracy can be achieved by re-meshing, however this is computationally expensive and requires careful judgement to ensure accuracy is reached whilst not wasting resources.

Numerous penetrations pass through the cylindrical BCU making the geometry non-axisymmetric. Therefore one-quarter of the unit has been analysed using a minimum of 40201 hexahedral finite elements. An existing parallel finite element (FE) code has been adapted for this project, allowing a more detailed analysis of the BCU to be carried out. The code can now handle pressure loads and self weight and can use one of two linear solvers (an iterative or a direct solver) to find a solution. Re-meshing schemes and error analysis have also been implemented for the BCU analyses.

These new tools have been used to carry out the analyses, looking at three different loading conditions for each of the four meshes. Conclusions have been drawn on the safety of the BCU under these conditions and the necessity for remeshing. Concrete peak stress envelopes for elastic analysis and principal stress plots have been used in post processing of the data, allowing the structural condition of the BCU to be investigated. Concrete behaviour is not very well understood under multi-axial loading conditions due to difficulty in obtaining laboratory data (particularly under combined tension and compression). Three different peak stress envelopes are considered and compared when carrying out elastic failure analysis of the BCU.

1.2 Project aims

The aims for the project are as follows: :

1. To make use of MPI to extend and run a parallel finite element program on a distributed computer (specifically using the Durham University's facilities)
2. To develop an existing parallel finitie element (FE) code (`ParaFEM`[20]) in order to analyse part of a nuclear reactor vessel (the BCU) subjected to extreme loading.
3. To understand how to use an advanced linear solver (a key component of FEA) called `MUMPS` and incorporate this into `ParaFEM`
4. To refine an existing BCU mesh, generating meshes containing higher order elements and more elements
5. To generate realistic loading files and boundary condition files for the BCU analysis
6. To run BCU analyses using different numbers of computer cores and compare runtime speeds.
7. To understand and then code the Z^2 error estimator for 3D hexahedral finite elements.
8. To obtain error estimations for the meshes used in the BCU analyses.
9. To make use of principal stress plots and peak stress envelopes to evaluate the FEA results and illustrate the differences between the various strength criteria employed.

1.3 Thesis breakdown

Chapters 2 and 3 provide a description of FEA theory and how FEA code can be run in parallel, specifically looking at the distributed computer available for research at Durham University. Chapters 4 and 5 contain a record of the most significant part of the project's contribution. Chapter 4 describes how `ParaFEM`, an existing parallel FEA code has been developed to incorporate features allowing the BCU to be analysed and the error in the analysis estimated. This section also describes a new mesh refinement strategy coded in Fortran 90, written to allow 8 and 20-noded hexahedral element meshes, such as the mesh available for the BCU, to be automatically refined. Chapter 5 describes the application for this work (the BCU), how the `ParaFEM` input files were generated and the FEA results obtained. Postprocessing results are also described where peak stress envelopes and principal stress plots have been used to get

an idea of the structural condition and safety of the BCU under different loading conditions. Finally in Chapter 6 the author's original contributions have been summarised, a series of conclusions are drawn from the work and several recommendations are made for possible future extensions to this study.

Chapter 2

Finite element analysis

Throughout the project Finite Element Method (FEM) has been used to carry out the stress analyses. Here some background information is provided on the theoretical basis of 3D linear FEM. The governing system of equations (the strong form) for standard structural stress analysis will be developed by considering three fundamental concepts used in continuum mechanics: equilibrium, constitutive relationships and strain-displacement relationships (see Sections 2.1, 2.2 and 2.3 respectively). In Section 2.4 the weak form is presented, allowing algebraic FEM equations to be developed which describe the behaviour of the solid by relating the nodal displacements to the nodal forces via the structural stiffness. In Section 2.5 information will be given on the 8 and 20-noded isoparametric hexahedral elements used in this project, and finally some attention is given to non-linear FEA, even though such analyses have not been carried out in this project.

2.1 Equilibrium

The strong form solution will be derived here by considering the force equilibrium of a small element (see Figure 2.1). Equilibrium describes the balance of internal and external forces acting on a body. Equilibrium of a 3D body can be written in terms of the stresses,

$$\{\sigma\} = \begin{Bmatrix} \sigma_{xx} \\ \sigma_{yy} \\ \sigma_{zz} \\ \sigma_{xy} \\ \sigma_{yz} \\ \sigma_{zx} \end{Bmatrix}, \tag{2.1}$$

and body forces per unit volume,

$$\{f_b\} \begin{Bmatrix} f_{b_x} \\ f_{b_y} \\ f_{b_z} \end{Bmatrix}. \tag{2.2}$$

The following expression may be written for equilibrium when considering the forces acting in the x

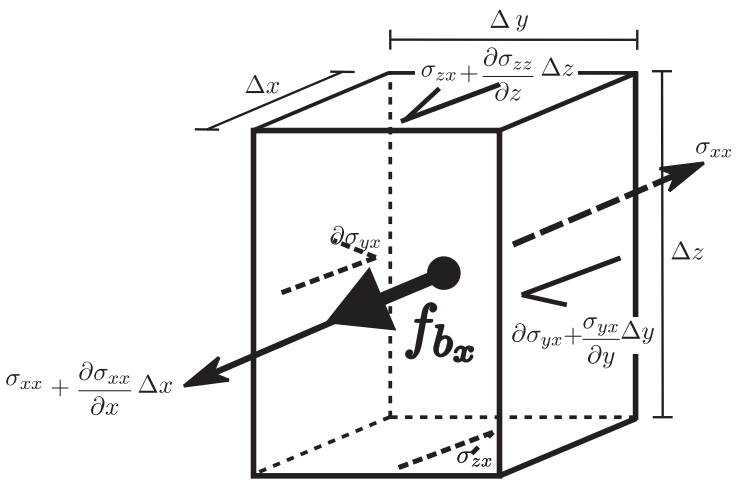


Figure 2.1: Stresses at a point in equilibrium - considering the x direction.

direction as shown in Figure 2.1

$$\begin{aligned}
&\sigma_{xx}\Delta y\Delta z+\frac{\sigma_{xx}}{\Delta x}\Delta x\Delta y\Delta z-\sigma_{xx}\Delta y\Delta z+\\
&\sigma_{yx}\Delta x\Delta z+\frac{\sigma_{yx}}{\Delta y}\Delta y\Delta x\Delta z-\sigma_{yx}\Delta x\Delta z+\\
&\sigma_{zx}\Delta x\Delta y+\frac{\sigma_{zx}}{\Delta z}\Delta z\Delta x\Delta y-\sigma_{zx}\Delta x\Delta y+f_{b_x}\Delta x\Delta y\Delta z=0.
\end{aligned}\tag{2.3}$$

Similar expressions can be written in the y and z directions. All three expressions simplify to form the following three equations

$$\begin{aligned}
&\frac{\partial\sigma_{xx}}{\partial x}+\frac{\partial\sigma_{yx}}{\partial y}+\frac{\partial\sigma_{zx}}{\partial z}+f_{b_x}=0,\\
&\frac{\partial\sigma_{yy}}{\partial y}+\frac{\partial\sigma_{xy}}{\partial x}+\frac{\partial\sigma_{zy}}{\partial z}+f_{b_y}=0 \qquad \text{and}\\
&\frac{\partial\sigma_{zz}}{\partial z}+\frac{\partial\sigma_{xz}}{\partial x}+\frac{\partial\sigma_{yz}}{\partial y}+f_{b_z}=0.
\end{aligned}\tag{2.4}$$

Given symmetry of the stress tensor ($\sigma_{xy}=\sigma_{yx}$ etc.), these three equations can be combined in a single matrix equation

$$\begin{bmatrix}
\frac{\partial}{\partial x} & 0 & 0 & \frac{\partial}{\partial y} & 0 & \frac{\partial}{\partial z}\\
0 & \frac{\partial}{\partial y} & 0 & \frac{\partial}{\partial x} & \frac{\partial}{\partial z} & 0\\
0 & 0 & \frac{\partial}{\partial z} & 0 & \frac{\partial}{\partial y} & \frac{\partial}{\partial x}
\end{bmatrix}
\begin{Bmatrix}
\sigma_{xx}\\ \sigma_{yy}\\ \sigma_{zz}\\ \sigma_{xy}\\ \sigma_{yz}\\ \sigma_{zx}
\end{Bmatrix}+
\begin{Bmatrix}
f_{b_x}\\ f_{b_y}\\ f_{b_z}
\end{Bmatrix}=
\begin{Bmatrix}
0\\0\\0
\end{Bmatrix}.\tag{2.5}$$

When (2.5) is written in a symbolic matrix notation it takes the following form

$$[L]^T\{\sigma\}+\{f_b\}=\{0\}.\tag{2.6}$$

This partial differential equation of equilibrium (together with the boundary conditions) is known as the strong form expression. The strong form will now be modified in the following sections through the

consideration of strain-displacement and constitutive relationships.

2.2 Constitutive relationships

Linear elastic constitutive models are used to describe the relationship between stresses and strains in a continuum during the deformation process. The constitutive relationship for all stress and strain components, independent of symmetries existing in the body, can be written in tensor notation as

$$\sigma_{ij} = D_{ijkl}\epsilon_{kl}. \tag{2.7}$$

D_{ijkl}, is the 4^{th} order constitutive stiffness tensor, containing 81 constants. D_{ijkl} maps the relationship between σ_{ij}, the 2^{nd} order stress tensor containing 9 stress components, and ϵ_{kl}, the 2^{nd} order strain tensor containing 9 strain components. The 81 constants in the constitutive stiffness tensor are called elastic constants. However, two symmetries exist in all constitutive tensors

1. $D_{ijkl} = D_{jikl}$ due to symmetry of the stress tensor ($\sigma_{ij} = \sigma_{ji}$) and

2. $D_{ijkl} = D_{ijlk}$ due to symmetry of the strain tensor ($\epsilon_{kl} = \epsilon_{lk}$).

These symmetries allow the constitutive tensor to be expressed as a 6 by 6 constitutive matrix, $[D]$, such that

$$\{\sigma\} = [D]\{\epsilon\}. \tag{2.8}$$

Using 2.8, the strong form solution, 2.6, can now be written as,

$$[L]^T [D]\{\epsilon\} + \{f_b\} = \{0\}. \tag{2.9}$$

$\{\sigma\}$ and $\{\epsilon\}$ are 6 by 1 column vectors containing the 6 unique stress and strain values respectively. This reduced form of constitutive relationship will be used in this thesis for the linear elastic behaviour.

2.3 Strain-displacement relationships

Deformation of a solid body is dependent on the rate of deformation (movement of material particles relative to each other). This property is called strain. Small strain takes account of relative fractional movements of particles and is described by the following six components

$$\{\epsilon\} = \begin{Bmatrix} \frac{\partial u}{\partial x} \\ \frac{\partial v}{\partial y} \\ \frac{\partial w}{\partial z} \\ \frac{\partial u}{\partial y} + \frac{\partial v}{\partial x} \\ \frac{\partial v}{\partial z} + \frac{\partial w}{\partial y} \\ \frac{\partial w}{\partial x} + \frac{\partial u}{\partial z} \end{Bmatrix}, \tag{2.10}$$

where u, v, and w are the components of displacement in the x, y and z directions respectively. Strain–displacement relationships can also be written in matrix notation

$$\left\{\epsilon\right\} = \begin{bmatrix} \frac{\partial}{\partial x} & 0 & 0 \\ 0 & \frac{\partial}{\partial y} & 0 \\ 0 & 0 & \frac{\partial}{\partial z} \\ \frac{\partial}{\partial y} & \frac{\partial}{\partial x} & 0 \\ 0 & \frac{\partial}{\partial z} & \frac{\partial}{\partial y} \\ \frac{\partial}{\partial z} & 0 & \frac{\partial}{\partial x} \end{bmatrix} \begin{Bmatrix} u \\ v \\ w \end{Bmatrix} \tag{2.11}$$

or, more compactly, as

$$\{\epsilon\} = [L]\{d\} \tag{2.12}$$

where $[L]$ is the differential operator. The strong form expression (2.6) can now be written as

$$[L]^T [D] [L] \{d\} + \{f_b\} = \{0\}. \tag{2.13}$$

2.4 Finite element equations

So far, only the strong form solution has been derived providing force equilibrium in terms of the spatial derivatives of displacement (2.13). This cannot be solved in its current form using the FEM therefore the strong form solution is adapted to derive the weak form solution. This derivation is carried out by pre–multiplying the equation by an arbitrary vector expressed in terms of the shape functions $[N]$ and integrating over the volume and applying the Gauss-Green theorem, to give

$$\int_\Omega \left([L][N]\right)^T [D][L][N]\, d\Omega \{d\} = \int_S [N]^T \{t\}\, dS + \int_\Omega [N]^T \{f_b\}\, d\Omega. \tag{2.14}$$

where $[N]$ is a matrix containing the shape functions associated with each node, such that for an 8-noded element

$$[N] = [N_1 [I] ... N_8 [I]], \tag{2.15}$$

where $[I]$ is the identity matrix. The weak form solution provides an equation which can be established by integrating over the body to form a stiffness matrix relating nodal displacements to nodal forces [7]. This stiffness matrix, $[K]$, is formed on the left hand side by integrating the expression numerically using Gaussian quadrature, considering contributions from each Gauss point (i = 1....number of Gauss points(nGp))

$$\underbrace{\sum_{i=1}^{i=nGp} [B]^T [D][B]\, \omega \det(J)}_{[K]} \{d\} = \{f_{ext}\} + \{f_b\}, \tag{2.16}$$

where $\det(J)$ is the determinant of the Jacobian matrix and w is the weight function associated with each Gauss point. The two terms on the right of (2.14) are expressed as $\{f_{ext}\}$ and $\{f_b\}$ and $[B]$ is the strain-displacement matrix containing the shape function derivative with respect to the global co-ordinate system. That is

$$[B] = [L][N] \tag{2.17}$$

where $[L]$ is a matrix of differential operators. The FE equilibrium equation for a single element is

$$[K^e]\{d^e\} = \{f^e\} \tag{2.18}$$

where $[K^e]$ is the element stiffness matrix, $\{d^e\}$ is the vector of element nodal displacements and $\{f^e\}$ is the element nodal force vector.

Traditionally FEM programs have been based on the assembly techniques where all the element $[K^e]$ matrices and $\{f^e\}$ vectors are assembled to form a global system of linear simultaneous equations for the complete structure

$$[K]\{d\} = \{f\}. \tag{2.19}$$

The weak form solution provides a linear system of equations which can then be solved either by iterative methods or by direct methods. Memory storage of the global system can present a problem in large meshes. Certain iterative techniques may offer advantages in this respect as they do not require the global system to be assembled therefore reducing the memory requirements for the problem.[1] Solution techniques for the linear system of equations (both iterative and direct) are discussed in Chapter 3.

2.5 Isoparametric hexahedral elments

The hexahedral elements used in this project are isoparametric. This implies that the same shape function are used to define the geometry as well as the variation in the displacements. In this section the global and local co-ordinate systems will be explained and the shape functions will be listed.

2.5.1 Local co-ordinate system and the co-ordinate mapping technique

The local and global domains are shown in Figure 2.2. The element can be defined within the local domain by defining the element boundaries using a local co-ordinate system ($\hat{\xi} = \pm 1$, $\hat{\eta} = \pm 1$ and $\hat{\zeta} = \pm 1$). It is possible to map the local domain to the global domain, which is defined in terms of the Cartesian co-ordinate system x, y and z, such that $x = x\left(\hat{\xi}, \hat{\eta}, \hat{\zeta}\right)$, $y = y\left(\hat{\xi}, \hat{\eta}, \hat{\zeta}\right)$ and $z = z\left(\hat{\xi}, \hat{\eta}, \hat{\zeta}\right)$ where every point identified by the local co-ordinate system has a unique point identified by the Cartesian co-ordinate system in the global domain. This mapping is carried out using the chain rule of differentiation given by

$$\begin{Bmatrix} dx \\ dy \\ dz \end{Bmatrix} = \begin{bmatrix} \frac{\delta x}{\delta \hat{\xi}} & \frac{\delta x}{\delta \hat{\eta}} & \frac{\delta x}{\delta \hat{\zeta}} \\ \frac{\delta y}{\delta \hat{\xi}} & \frac{\delta y}{\delta \hat{\eta}} & \frac{\delta y}{\delta \hat{\zeta}} \\ \frac{\delta z}{\delta \hat{\xi}} & \frac{\delta z}{\delta \hat{\eta}} & \frac{\delta z}{\delta \hat{\zeta}} \end{bmatrix} \begin{Bmatrix} d\hat{\xi} \\ d\hat{\eta} \\ d\hat{\zeta} \end{Bmatrix},$$

or

$$\begin{Bmatrix} dx \\ dy \\ dz \end{Bmatrix} = [J] \begin{Bmatrix} d\hat{\xi} \\ d\hat{\eta} \\ d\hat{\zeta} \end{Bmatrix},$$

where $[J]$ is the Jacobian matrix (a matrix containing first order partial derivatives of the global co-ordinate system with respect to the local co-ordinate system). This matrix allows the local co-ordinate

[1] Direct solvers also can be constructed in an element-by-element form.

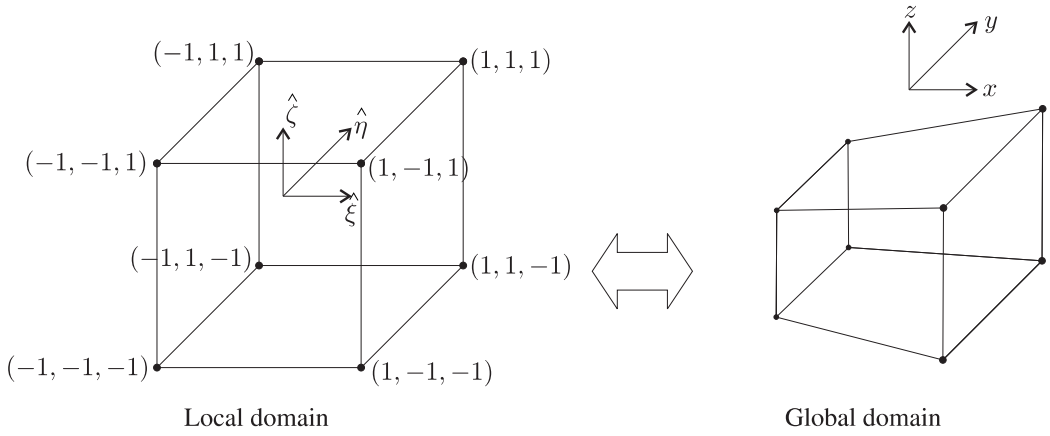


Figure 2.2: Mapping from the local to the global domain

system to be mapped to the global co-ordinate system and vice versa. The determinant of the Jacobian ($\det J$) must be non zero in order for a unique mapping to exist. This ensures the system of elements is stable which is one of the requirements needed for convergence towards a solution. This can be achieved by using a consistent node ordering pattern. Throughout this project a clockwise ordering of nodes has been used starting at node 1, when viewing an element face from the outside of the element, as shown in Figure 2.3. $[J]$ is used to calculate $[K]$ (as seen in Section 2.4), so that the variables and domain with respect to which the integration is made are transformed.

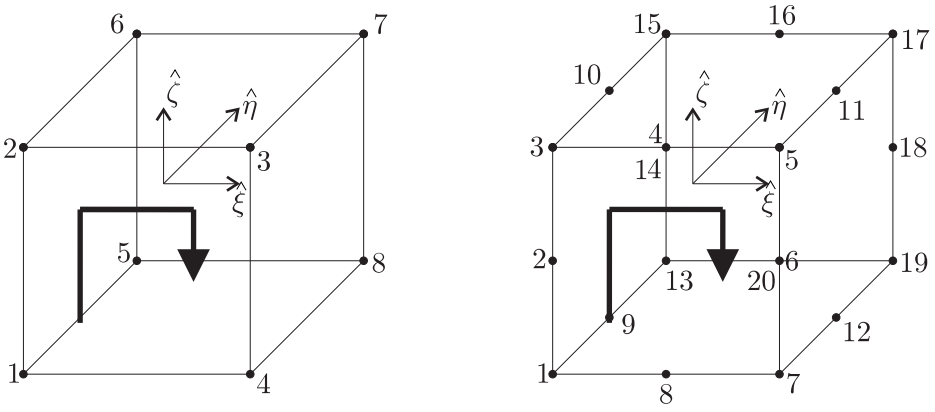


Figure 2.3: Node numbering for 8 and 20-noded element shape functions

2.5.2 Shape functions and their derivatives

Shape functions describe the geometry and displacements of an element. In this way the location of any point in an element can be calculated by knowing the local co-ordinates of the point, the shape functions for the element and the nodal displacements. The following equation illustrates this relationship for an 8-noded element,

$$\begin{Bmatrix} x \\ y \\ z \end{Bmatrix} = [N] \begin{Bmatrix} x_1 & y_1 & z_1 & . & . & . & x_8 & y_8 & z_8 \end{Bmatrix}^T, \tag{2.20}$$

where $[N]$ (see 2.15) contains the shape functions for nodes 1 to 8. Throughout this project both 8 and 20-noded hexahedral isoparametric elements have been used. The 8-noded element uses tri-linear shape functions, with either a single point reduced integration scheme or an eight point full integration scheme. The 20-noded element uses tri-quadratic shape functions, with either an eight point reduced integration scheme or a twenty-seven point full integration scheme. Both sets of shape functions are available within `ParaFEM` and can be specified within the input file `p121.dat`. Because 8-noded elements employ linear shape functions, they generally provide a less accurate solution compared to that provided by 20-noded elements where quadratic shape functions are employed, however when 20-noded elements are used the size of problem increases requiring more time and memory to solve. For each node there is an associated shape function. The nodes are identified using a consistent node numbering scheme shown in Figure 2.3. The 8-noded element shape functions are as follows

$$\begin{aligned}
N_1 &= \frac{1}{8}\left(1-\hat{\xi}\right)(1-\hat{\eta})\left(1-\hat{\zeta}\right) & N_2 &= \frac{1}{8}\left(1-\hat{\xi}\right)(1-\hat{\eta})\left(1+\hat{\zeta}\right) \\
N_3 &= \frac{1}{8}\left(1+\hat{\xi}\right)(1-\hat{\eta})\left(1+\hat{\zeta}\right) & N_4 &= \frac{1}{8}\left(1+\hat{\xi}\right)(1-\hat{\eta})\left(1-\hat{\zeta}\right) \\
N_5 &= \frac{1}{8}\left(1-\hat{\xi}\right)(1+\hat{\eta})\left(1-\hat{\zeta}\right) & N_6 &= \frac{1}{8}\left(1-\hat{\xi}\right)(1+\hat{\eta})\left(1+\hat{\zeta}\right) \\
N_7 &= \frac{1}{8}\left(1+\hat{\xi}\right)(1+\hat{\eta})\left(1+\hat{\zeta}\right) & N_8 &= \frac{1}{8}\left(1+\hat{\xi}\right)(1+\hat{\eta})\left(1-\hat{\zeta}\right)
\end{aligned} \tag{2.21}$$

and the 20-noded element shape functions are as follows

$$
\begin{aligned}
N_1 &= \frac{1}{8}\left(1-\hat{\xi}\right)\left(1-\hat{\eta}\right)\left(1-\hat{\zeta}\right)\left(-\hat{\xi}-\hat{\eta}-\hat{\zeta}-2\right) & N_2 &= \frac{1}{4}\left(1-\hat{\xi}\right)\left(1-\hat{\eta}\right)\left(1-\hat{\zeta}^2\right)\\
N_3 &= \frac{1}{8}\left(1-\hat{\xi}\right)\left(1-\hat{\eta}\right)\left(1+\hat{\zeta}\right)\left(-\hat{\xi}-\hat{\eta}+\hat{\zeta}-2\right) & N_4 &= \frac{1}{4}\left(1-\hat{\xi}^2\right)\left(1-\hat{\eta}\right)\left(1+\hat{\zeta}\right)\\
N_5 &= \frac{1}{8}\left(1+\hat{\xi}\right)\left(1-\hat{\eta}\right)\left(1+\hat{\zeta}\right)\left(\hat{\xi}-\hat{\eta}+\hat{\zeta}-2\right) & N_6 &= \frac{1}{4}\left(1+\hat{\xi}\right)\left(1-\hat{\eta}\right)\left(1-\hat{\zeta}^2\right)\\
N_7 &= \frac{1}{8}\left(1+\hat{\xi}\right)\left(1-\hat{\eta}\right)\left(1-\hat{\zeta}\right)\left(\hat{\xi}-\hat{\eta}-\hat{\zeta}-2\right) & N_8 &= \frac{1}{4}\left(1-\hat{\xi}^2\right)\left(1-\hat{\eta}\right)\left(1-\hat{\zeta}\right)\\
N_9 &= \frac{1}{4}\left(1-\hat{\xi}\right)\left(1-\hat{\eta}^2\right)\left(1-\hat{\zeta}\right) & N_{10} &= \frac{1}{4}\left(1-\hat{\xi}\right)\left(1-\hat{\eta}^2\right)\left(1+\hat{\zeta}\right)\\
N_{11} &= \frac{1}{4}\left(1+\hat{\xi}\right)\left(1-\hat{\eta}^2\right)\left(1+\hat{\zeta}\right) & N_{12} &= \frac{1}{4}\left(1+\hat{\xi}\right)\left(1-\hat{\eta}^2\right)\left(1-\hat{\zeta}\right)\\
N_{13} &= \frac{1}{8}\left(1-\hat{\xi}\right)\left(1+\hat{\eta}\right)\left(1-\hat{\zeta}\right)\left(-\hat{\xi}+\hat{\eta}-\hat{\zeta}-2\right) & N_{14} &= \frac{1}{4}\left(1-\hat{\xi}\right)\left(1+\hat{\eta}\right)\left(1-\hat{\zeta}^2\right)\\
N_{15} &= \frac{1}{8}\left(1-\hat{\xi}\right)\left(1+\hat{\eta}\right)\left(1+\hat{\zeta}\right)\left(-\hat{\xi}+\hat{\eta}+\hat{\zeta}-2\right) & N_{16} &= \frac{1}{4}\left(1-\hat{\xi}^2\right)\left(1+\hat{\eta}\right)\left(1+\hat{\zeta}\right)\\
N_{17} &= \frac{1}{8}\left(1+\hat{\xi}\right)\left(1+\hat{\eta}\right)\left(1+\hat{\zeta}\right)\left(\hat{\xi}+\hat{\eta}+\hat{\zeta}-2\right) & N_{18} &= \frac{1}{4}\left(1+\hat{\xi}\right)\left(1+\hat{\eta}\right)\left(1-\hat{\zeta}^2\right)\\
N_{19} &= \frac{1}{8}\left(1+\hat{\xi}\right)\left(1+\hat{\eta}\right)\left(1-\hat{\zeta}\right)\left(\hat{\xi}+\hat{\eta}-\hat{\zeta}-2\right) & N_{20} &= \frac{1}{4}\left(1-\hat{\xi}^2\right)\left(1+\hat{\eta}\right)\left(1-\hat{\zeta}\right). \qquad (2.22)
\end{aligned}
$$

2.6 Non-linear finite element analysis

Throughout this project the behaviour of the BCU has been restricted to the linear case, however real concrete structures exhibit non-linear responses under loading. We will therefore briefly discuss concrete's generalised non-linear response under multi-axial loading and how our linear assumptions could affect our results. A short summary will also be given on how to incorporate non-linear analysis into a FEA code and an example pseudo-code presented. Idealised one-dimensional stress-stain curves for tension and compression are shown in Figure 2.4. In compression, concrete initially behaves linearly elastically and then some non-linearity appears. Depending on the degree of compressive confinement present the material may exhibit a ductile response as the peak is attained. Thereafter softening occurs before reaching a point when no more load can be carried by the material. Concrete behaves quite differently under tension. Initially the behaviour is linear before showing a brittle softening response. The uniaxial tensile strength (f_t) of concrete is generally between $10-20\%$ of the uniaxial compressive strength (f_c). The strength of concrete under multi-axial loading is more complicated. A combination of all stresses must be considered since simple limitations of tensile and compressive uniaxial strength do not accurately predict the strength. For example, if a concrete specimen were subject to tensile loading in one direction and compressive loading in another then the stress state may be within the uniaxial tensile and compressive strength limits yet the specimen could still fail.

Non-linear FEA has not been used during this work, however it would be useful to incorporate non-linear analysis into the code to develop the work further. The pseudo- code which follows this description has been used to show how a non-linear algorithm might look (the numbers on the left hand side of the code represent the line numbers and are not part of the code, they have been used to reference lines of code as it is described). During non-linear analysis the solution is determined in a number of steps by applying the external force in a series of discrete load steps (line (9)). Before the external force is

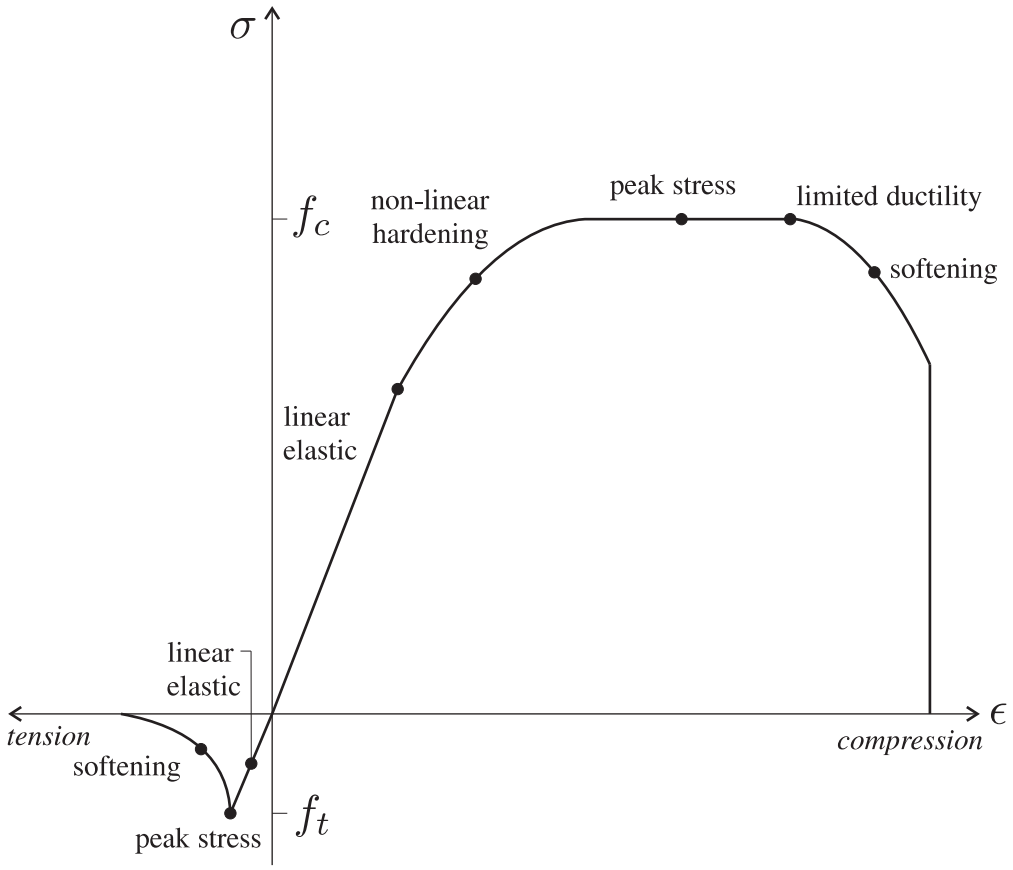


Figure 2.4: Idealised stress-strain curves for concrete under one-dimensional loading

applied the elastic stiffness matrix `[K]` is calculated (lines (2)-(7)). The loads steps are then applied and a solution approached (lines (9)-(35)). For each load step (applied in line (9)) an iterative (typically Newton-Raphson) procedure is carried out until the out of balance force `Df` is less than a given tolerance. This iterative procedure is shown in lines (13)-(26). `Df` is calculated by finding the difference between the externally applied force vector `f_ext` and the internal force vector `f_int` (line (11) initially and line (27) during each iteration). During each iteration a new `f_int` and stiffness matrix `[K]` are calculated (lines (13)-(26)). This procedure is carried out until `Df` is less than a given tolerance, or the maximum number of permitted iterations is reached. At this point the estimated displacement vector `d_old` is reassigned (line (29)) and the stress vector can then be calculated (lines (30)-(34)). The pseudo-code is shown below:

```
(1)   {d_old}={0}, {d_new}={0}, {sig_old}={0}, {f_int}={0}, ||Df||=2xtol
(2)   do elem=1:nelm
(3)     do gp=1:ngp
(4)       [Ke]=sum([B]^T[De][B]w|J|
(5)     end
(6)     [Ke]-->[K]
(7)   end
(8)
(9)   do loadstep=1:nloadstep
```

```
(10)     read in {f_ext}
(11)     {Df}={f_ext}-{f_int}
(12)     NRit=0
(13)     while(||Df||>tol & NRit<maxNRit)
(14)       NRit=NRit+1
(15)       [K]{Dd}={Df}
(16)       {d_new}={d_old}+{Dd}
(17)       do elem=1:nelm
(18)         do gp=1:ngp
(19)           {Deps}=[B]{{d_new}-{d_old}}
(20)           {Deps}, {sig_old}-->constitutive model-->{Dsig}, [Dep] or [De]
(21)           [Ke]=sum([B]^T[Dep][B]w|J|
(22)           {fe_int}^T=sum{{sig_old}+{Dsig}}^T[B]w|J|
(23)         end
(24)         {fe_int}-->{f_int}
(25)         [Ke]-->[K]
(26)       end
(27)       {Df}={f_ext}-{f_int}
(28)     end
(29)     {d_old}={d_new}
(30)     do elem=1:nelem
(31)       do gp=1:ngp
(32)         {sig_new}={sig_old}+{Dsig}
(33)       end
(34)     end
(35)   end
```

Chapter 3

Parallel finite element analysis

In this section, parallel computing, the Message Passing Interface (MPI) and the high performance computing (HPC) service at Durham University will be described. `ParaFEM`[20], the existing parallelised Fortran 90 FEA code will be introduced and two alternative parallel linear solvers will be discussed.

3.1 Parallel computing

In order to understand the mechanics of parallel programming it is important to be clear on terminology. Often literature and instruction manuals can confuse the matter by interchanging terminology. Common definitions for a computer cluster, node, processor, core, process, memory bus and thread have been listed here and then used to mean the following throughout the thesis.

- **Computer Cluster**: A set of computers, usually physically located near one another and connected to one another in order to solve computing problems more efficiently. These networks of computers are often referred to as **HPC services**, **super-computers** or **distributed memory environments**.
- **Node**: A piece of computer hardware connected to and communicating with other nodes within a computer cluster. Each node runs its own operating system, and can consist of one or several processors. Within a cluster, nodes include compute nodes, I/O nodes, service nodes and network nodes and special-purpose nodes.
- **Processor**: A computing component able to read and execute a program. Processors are made up of **cores**. Processors can be single or multi-cored.
- **Core**: A sub-component of a processor. Each core can read and execute programs separately.
- **Process**: A process is the reading and execution of programming instructions. One process can be run on each **core**.
- **Memory Bus**: Subsystem involved in data transfer between processors.

- **Thread**: A thread is a subdivision of a process. A core can execute threads within a process simultaneously without them interfering with each other, for example a process might be separated into threads so that one thread carries out the task, another calculates the error and a third sends an error message to the user. This process is known as **multi-threading**. `OpenMP` can be used to make jobs run in parllel using multi-threading, however multi-threading has not be used in this project.

Parallel computing potentially speeds up calculations by performing multiple operations simultaneously. These operations would otherwise be carried out one at a time by a serial program. Parallel programming, as used in this project, reduces the computational effort by sharing the work load between cores. This reduction in computational effort must outweigh the time spent during inter–processor information sharing in order to have a net time saving. Parallel programming may also alleviate memory requirement if a distributed memory environment is being used where different processors each with their own memory space stores part of the program's data. There are two models available for parallel computing where the parallelism is specified by the programmer;

- **Shared memory model**: Multiple cores share access to a global memory space allowing them to exchange or share data efficiently, shown in Figure 3.1. The number of cores sharing a single memory space is limited by the bandwidth of the memory bus connecting the cores. This is generally in the region of $2 - 16$ cores.

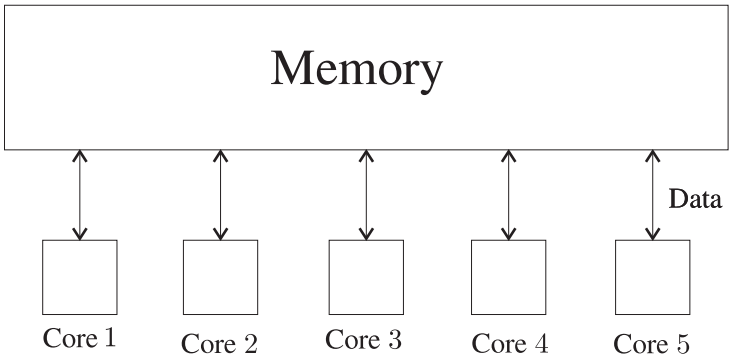


Figure 3.1: Shared memory model

- **Message passing model**: Multiple processors each with their own local memory communicate with each other by passing messages, shown in Figure 3.2. Message passing between two memories requires both processors to pass a message. MPI is a specific implementation of the message passing model and will be discussed in more detail in the next section.

Computer clusters are typically made up of a combination of the two models. Each processor contains multiple cores which share a memory (shared memory model). The processors are interlinked in a network (message passing model) to provide the complete service.

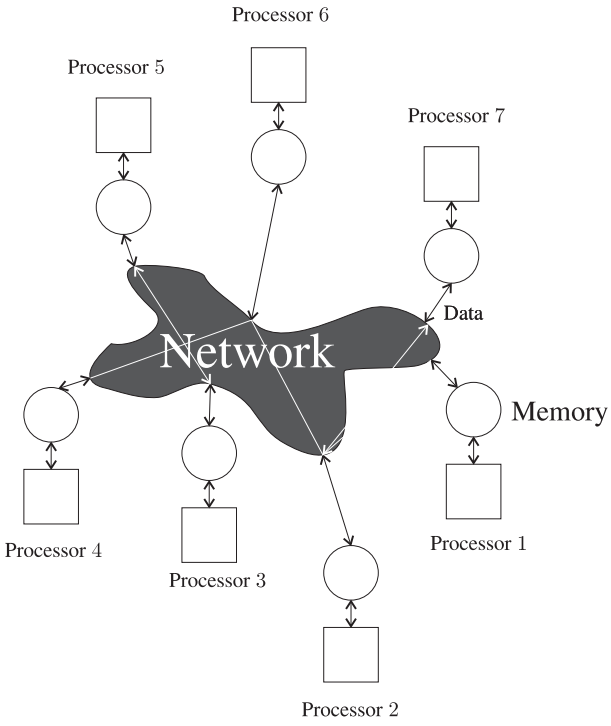


Figure 3.2: Message passing model

3.1.1 Message Passing Interface (MPI)

MPI is a language independent communications protocol for parallel computing. It is a library of function calls that co-ordinate the program as it runs on multiple cores in a distributed memory environment. Each time data is communicated between cores (i.e. data is copied from one memory to another if it is not shared) MPI calls are made by all cores involved. Communication is always cooperative, which means that it can only be executed fully if the first core executes a *send* function and the second core executes a *receive* function. For such functions to be operational, certain arguments must be provided by the core calling them, for example: the address, size and type of data to be communicated and the core involved in the communication (destination and source core). A *tag* or matching integer is required as an argument to ensure the co-operative nature of communication. The *tag* ensures that the *send* function is only deemed complete once the *receive* operation capable of identifying data containing that *tag* has been completed. The structure of a typical MPI program will now be described using a piece of pseudo-code. An explanation for each line will be given in the next paragraph (the numbers on the left hand side of the code represent the line numbers and are not part of the code, they have been used to reference lines of code as it is described). The pseudo-code is shown below:

```
1 program program name
2 include 'mpif.h'

–variable declaration–

11 call MPI_INIT(ierr)
12 call MPI_COMM_RANK(MPI_COMM_WORLD, myid, ierr)
```

```
13 call MPI_COMM_SIZE(MPI_COMM_WORLD, numpe, ierr)
```

–main program–

```
101 call MPI_FINALIZE(ierr)
102 program end
```

The program includes starting and finishing the program (lines 1 and 102), declaring the library files (line 2) and four MPI calls. `MPI_INIT(ierr)` (line 11) must be the first MPI subroutine called within any MPI code. It establishes the MPI environment and allows other MPI functions to be called. `ierr` (lines 12, 13 and 101) is the error code argument and is the last argument of every MPI subroutine, it returns either `MPI_SUCCESS` or an appropriate error code. `MPI_COMM_SIZE` (line 13) returns the number of cores/number of processes being run (`numpe`). This is defined by the user when the program is executed. By calling `MPI_COMM_RANK` (line 12) each core can find its rank (the core identification number used for each communication) in the group associated with the communicator. `MPI_FINALIZE` (line 101) is the last MPI subroutine to be called and it closes down the MPI environment. `MPI_INIT(ierr)` and `MPI_FINALIZE(ierr)` can only be called once in any program.

Here is an example `hello_world.f90` Fortran 90 program where MPI messages are passed between cores in the main body of the program:

```
1 program hello_world
2 include 'mpif.h'
3 integer rank, size, ierr, tag, status(MPI_STATUS_SIZE)
4 character(12) message
5
6 call MPI_INIT(ierr)
7 call MPI_COMM_SIZE(MPI_COMM_WORLD, size, ierr)
8 call MPI_COMM_RANK(MPI_COMM_WORLD, rank, ierr)
9 tag = 100
10
11 if(rank .eq. 0) then
12   message='hello'
13   do i=1, size-1
14     call MPI_SEND(message, 12, MPI_CHARACTER, i, tag, MPI_COMM_WORLD, ierr)
15   end do
16 else
17   call MPI_RECV(message, 12, MPI_CHARACTER, 0, tag, MPI_COMM_WORLD, status, ierr)
18 end if
19 print(*,*) 'node', rank, ':', message
20
21 call MPI_FINALIZE(ierr)
22 end program hello_world
```

Lines 11-19 are the main body of the program. Each core is running the program but the program specifies

different cores to do different things. Lines $11-15$ specifies the core of rank 0 must send the message to all the other cores (cores of rank 1 to size-1). Lines $15-18$ ask all cores bar the core of rank 0 to receive the message sent. The message is then printed by all cores (line 19).

This program gives examples of several MPI functions where the function requires several arguments. For example `MPI_SEND` has seven arguments. The `message` is the character data being sent, `12` is the number of data elements of the specified type to be sent, `MPI_CHARACTER` is the type of data, `i` is the destination of the data, `tag` is the tag given to the message, `MPI_COMM_WORLD` is the group of cores communicating and `ierr` is the error message.

MPI parallel programs have to be complied and linked with MPI libraries `mpif.h` (compiled by the same compiler). The MPI compiler `mpif90` ensures the MPI libraries are linked successfully with the Fortran 90 compiler. MPI programs can then be executed interactively from the command line or by using a batch queue system.

3.1.2 The high performance computing service in Durham

Durham University's HPC service is a computer cluster known by the name *Hamilton*. Such a service enables researchers to verify, validate and execute parallel programs. *Hamilton* consists of a 228 node (1824 processor core) SuSE Linux Cluster. Each node comprises of two quad-core Intel Xeon E5520/2.26 GHz Nehalem processors (total of eight cores per node), 24 GB Memory, 160 GB disk space and a QDR InfiniBand interconnection. Throughout this project *Hamilton* has been used, initially in an interactive mode to test and debug the code, and then in batch mode to run the code on up to sixteen cores. *Hamilton* allows parallel code to run on up to 128 cores but the `ParaFEM` code used in this project is hardwired to be run on up to sixteen cores.

Access to *Hamilton* is provided via a Durham University CIS Linux service from within the University network. It is possible to connect to *Hamilton* from outside the university network by connecting to another public university computer, for example *Vega*. Connecting to *Hamilton* from a Windows PC requires `PuTTY`, a client which runs on a PC under Microsoft Windows and allows access to Linux machines remotely using Secure Shell (`ssh`). `ssh` is a network protocol that allows data to be exchanged between two network devices via a secure channel. `PuTTY` is run via the command line, however a graphical interface can be provided by VCN Viewer. This viewer allows the user to use a windowing environment whilst connected to a Linux desktop computer enabled by a Gnome Desktop environment. Once connected to a Durham University Linux machine, a connection can be made directly to *Hamilton* via the command line. The required ssh command is **ssh hamilton-new**.

Detailed instructions on how to use `PuTTY` and `VNC Viewer` to connect to *Vega* and *Hamilton* from a remote Windows PC can be found by authorised users on the Durham University HPC web pages [6].

Hamilton is a complex computer system and therefore requires a modular environment to allow users to set up their own user environment in order to find the required executable and libraries for their usage. Environment modules provide a way of selectively activating and deactivating the complete environment to allow particular packages and versions of library files to be found. The commands required to set up a unique user environment are:

`module`	print usage instructions
`module av`	prints modules available
`module load <module name>`	add a module to user's environment
`module unload <module name>`	removes a module from user's environment
`module purge`	removes all modules from user's environment
`module list`	lists all modules loaded by user

The following modules were required for the work carried out in this thesis:

`openmpi/intel/64/1.4.2`	Provides access to an Intel Fortran 90 compiler and Intel compiled MPI libraries
`intel/mkl/10.1.1`	Provides access to Intel Compiled BLAS, BLACS and ScaLAPACK libaries within the Math Kernel Library (`mkl`)
`sge/current`	Provides access to the *Grid Engine*, e.g. allows use of queueing commands: `qrsh`, `qsub` etc.

Hamilton controls the job (the name given to the execution of a program) using a queuing system called *Grid Engine*. This is a software package made available by Oracle[19]. This system functions to manage work on behalf of the user and efficiently manage the resources of the system. Jobs will be held until the correct number of cores become available. On *Hamilton*, the correct way to execute a parallel job is by using the batch queue system, however special nodes have been reserved for debugging code where code can be executed interactively using up to sixteen cores. These nodes can be accessed by the queuing system via the `qrsh` command. Once logged into one of these allocated nodes the following command can be typed into the command line, `mpirun -np number_of_processes ./executable` where `np` is the number of processes the user wishes to run. Alternatively, a batch file can be written and then submitted to a specified queue. Templates are provided by Durham University's Computer Information Service (CIS). Users can then monitor the status of their job using the `qstat` command.

To protect the system, users' Durham University network drives are not mapped to their Hamilton directory where each user has their own memory allocated to them. Files must be copied across the secure channel using the `ssh` copy command (`scp`). The copy command is,

`scp` *`filename_and_location destination`*

for example,

```
scp p121.f90 vega:./MSc_work/bcu_analyses
```

which copies `p121.f90` from the local directory on Hamilton to the folder `bcu_analyses` on *Vega*.

3.2 Parallel finite element analysis

When using FEA to analyse large structures, computational effort and memory requirements can be high due to the quantity of input data and the size of the system of equations. A significant percentage of the total runtime for large structures is taken up in solving the system of linear equations. As described in Chapter 2, the central equation in displacement based FEA stress analysis is $[K]\{d\} = \{f\}$ (2.19).

This equation must be solved to find the displacement vector $\{d\}$. Due to the size of the system, $[K]$ cannot easily be inverted, so $\{d\}$ has to be solved for using a linear solver. The efficient running of the linear solver is vital for the efficiency of the code as a whole. In this section we will focus on parallel linear solvers, in particular looking at the implementation of a direct solver into an existing parallel finite element analysis (PFEA) code called `ParaFEM`.

3.2.1 `ParaFEM`

`ParaFEM` [20] is a freely available parallel FEA code, downloaded from the `ParaFEM` website. The subroutines are written in Fortran 90 making use of MPI. The code was developed from existing software created by Lee Margetts [16] under the supervision of I.M. Smith and M.A. Pettipher at the University of Manchester. That work was an extension of serial codes written by I.M. Smith and D.V. Griffiths. A description of the `ParaFEM` software and its functionalities is provided in Chapter 12 of Smith and Griffiths book: *Programming the Finite Element Method* [22]. `ParaFEM` is built by assembling subroutines, grouped in modules, from within the Smith and Griffiths' FE library. `ParaFEM` subroutines and FEA code structure have been used as the building blocks and provide the backbone for the parallel analysis carried out in this project.

The original `ParaFEM` program `p121.f90` uses fourteen modules containing subroutines. To compile `p121.f90` quickly, with all the modules, a makefile was written. The module files `*.mod`, main program `p121.f90` and makefile `makefile.mk` were placed in the same folder. To run the makefile, the following command was used, `make -f makefile.mk`. This compiled all the modules and the main program and outputted an executable `p121_exe`. The program could then be run using a batch script or interactively, see Section 3.1.1.

ParaFEM requires four input files to run. An additional input file has also been added to provide pressure data inputed into the FEA model. Table 3.1 summarises the contents of each input file and further information on the format and contents of each input file can be found in Appendix A (along with examples of all the input files and the makefile for `ParaFEM`).

File	Contents
`p121.dat`	Basic control data
`p121.d`	The geometry of the problem
`p121.bnd`	The boundary conditions
`p121.lds`	Nodal loads to be applied
`p121.prs`	Pressure loads to be applied

Table 3.1: `ParaFEM` input file summary

The program is split into 14 sections and each section is given a title (existing in the code as a comment, providing helpful markers to navigate the program). Sections 1 and 2 declare variables used in the main program and the dynamic arrays - these have to be allocated memory, however this is done later in the program only when the arrays need to be used. In the third section the job name, control data, mesh data, boundary conditions and loading conditions are read from the input files specified in 3.1. The dynamic arrays are then allocated memory space in Section 4. Section 5 then loops through the elements to find the steering array (the node numbers for each element listed in the correct order) and the number of equations

to be solved. The inter-processor communication tables are then formed in Section 6. More arrays (this time arrays local to each processor) are allocated memory space in Section 7, this allocation is dependent on the number of equations that each processor has to solve. In Section 8 the element stiffness matrices are calculated and stored. Each processor calculates and stores only the element stiffness matrices required to solve the equations it has been specified. The diagonal pre-conditioner is formed in Section 9. The force vector is organised in Section 10 and then the PCCG solver carries out the iterative process of finding the estimation to the nodal displacements in Section 11. Finally in Sections 12, 13 and 14 the stresses and nodal displacements are collected back on the host processor and the run-time performance data is outputted.

3.2.2 Linear solvers

The equilibrium equations (2.19) can be solved either by an iterative method or by a direct method. `ParaFEM` uses a diagonally pre-conditioned conjugate gradient (PCCG) solver, which is an iterative solver. This element–by–element approach avoids the need to assemble the global stiffness matrix. ParaFEM's `p121` Fortran 90/MPI program was modified by the author such that `MUMPS` could be used instead of the conjugate gradient solver. `MUMPS'` direct solver adopts a multi frontal strategy, having the potential to solve larger problems more quickly. In this section both solvers will be described. The implementation of the `MUMPS` package on Durham's HPC service will be explained, and integration of this solver into the existing Fortran code will be explained by identifying necessary calls and parameter settings. The code is also shown in Appendix C.

3.2.3 Pre-conditioned conjugate gradient solver (PCCG)

The PCCG method[22] is an iterative technique where the algorithm generates a sequence of improving approximate solutions, until it lies within a given tolerance. A description, as found in Smith Giffiths book[22], will be given here.

An initial guess is made for the vector $\{U\}$, this guess is called the initial trial solution ($\{U\}_0$). Then the residual or error $\{R_0\}$ is calculated for the trial solution

$$\{P_0\} = \{R_0\} = \{F\} - [K_m]\{U\}_0. \tag{3.1}$$

Here we are considering the finite element equation Equation 2.19, therefore $\{U\}$ is the displacement vector, $\{F\}$ is the force vector and $[K]$ is the stiffness matrix. The following k steps (see 3.2)[22] are carried out until the difference between the displacement calculated in two consecutive steps $\{U\}_{k+1}$ and $\{U\}_k$ is smaller than a specified tolerance (in `ParaFEM` this tolerance is specified in the input file `p121.d`

- see Table 3.1).

$$\begin{aligned}
\{Q\}_k &= [K]\{P\}_k \\
\hat{\alpha}_k &= \frac{\{R\}_k^T\{R\}_k}{\{P\}_k^T\{Q\}_k} \\
\{U\}_{k+1} &= \{U\}_k + \alpha_k\{P\}_k \\
\{R\}_{k+1} &= \{R\}_k - \alpha_k\{U\}_k \\
\hat{\beta}_k &= \frac{\{R\}_{k+1}^T\{R\}_{k+1}}{\{R\}_k^T\{R\}_k} \\
\{P\}_{k+1} &= \{R\}_{k+1} + \beta_k\{R\}_k
\end{aligned} \tag{3.2}$$

Note that $\{Q\}, \{P\}$ and $\{R\}$ are vectors, with a length equal to the number of equations which need to be solved and $\hat{\alpha}$ and $\hat{\beta}$ are scalars. This process can be carried out element by element without assembling $[K]$, such that $\{Q\} = \Sigma_{i=1}^{nels}[K^e]_i\{p\}_i$ where $[K^e]_i$ is the stiffness matrix of the i^{th} element, $nels$ is the number of elements and $\{p\}_i$ is the appropriate part of $\{P\}$.

Preconditioning is used to accelerate convergence. The FE equation is pre-multiplied by the 'pre-conditioner' matrix $[P]$ such that

$$[P][K_m]\{U\} = [P]\{F\}. \tag{3.3}$$

If $[K_m]^{-1}$ could be calculated the solution could be obtained in one step

$$\{U\} = [K_m]^{-1}\{F\}. \tag{3.4}$$

However this is not computationally possible for larger matrixes therefore relatively crude approximations to $[K_m]^{-1}$ can be used to construct $[P]$ and then be used in the iterative process. For example, diagonal pre-conditioning uses the inverse of the diagonal terms of $[K_m]$ as a matrix $[P]$. The PCCG solver within `ParaFEM` uses the simplest form of diagonal pre-conditioning [22].

3.2.4 MUMPS

`MUMPS` [2] [3] [18] is a package for solving systems of linear equations using a direct method based on a multi-frontal approach. The parallel version of `MUMPS` makes use of `BLAS`, `BLACS` and `ScaLAPACK` libraries [18]. These libraries contain linear algebra sub-programs capable of carrying out operations such as vector and matrix operations. These libraries are available on the *Hamilton* cluster within the `mkl-intel` libraries and access can be provided by specifying the correct path location within the `make` file and using the correct modules `openmpi/intel/64/1.4.2` and `intel/mkl/10.1.1`

`MUMPS` uses the direct method to solve the linear system of equations. This method obtains a solution using Gaussian elimination. `MUMPS` uses a frontal system. The main idea of a frontal system is to assemble equations and eliminate the variables (using Gaussian elimination) at the same time, rather than assembling the whole matrix then solving the whole system[10]. As soon as all contributions to the stiffness matrix for a given degree of freedom within a finite element mesh have been accumulated the degree of freedom is eliminated from the system using standard Gaussian operations. The degrees

of freedom currently being worked on can be visualised as a front. This area of the stiffness matrix is a transition region between the part of the system solved for and the part not touched yet. The whole stiffness matrix is never assemble in memory, instead assembly and elimination are alternated with the factors written to disk as the process is carried out. Re-ordering of the stiffness matrix is crucial to the speed of the the process. `MUMPS` is also a multi-frontal solver. This means it uses several fronts at the same time to solve the system of equations. This means several processors can be used each working on their own front.

Running `MUMPS` as part of an existing `Fortran 90` code

`MUMPS` is able to solve linear systems of equations either in assembled, elemental or distributed assembled form. These forms indicate how the stiffness matrix is stored on the cores. Assembled form means that the stiffness matrix is initially assembled on the host core, elemental form means that the stiffness matrix is inputted in terms of individual element stiffness matrices on the host core, whilst distributed assembled form means the stiffness matrix is assembled across the cores, each core storing the part of the stiffness matrix it will solve. For each of the three problem types a set of `MUMPS` variables have to be defined on either the host processor or the local processors. Within `ParaFEM` the system of linear equations can be solved using all three methods. All three have been successfully implemented providing identical displacement outputs to at least six significant figures when solving simple unit cube problems and also the BCU (both of these implementations have been explained in Chapter 4). However a distributed assembled matrix input was used for the BCU FEAs because this method allowed a larger system of equations to be solved without running into memory allocation problems. The variables required to run a distributed assembled problem are listed in Table 3.2. The stiffness matrix is defined by `MUMPS` matrix [A], and the force vector is defined by `MUMPS` array RHS. The solution is then stored in RHS once the system has been solved. To implement `MUMPS` within `ParaFEM` such that it can solve the problem as a distributed assembled problem the following steps had to be taken;

1. Include new library files
   add lines:

   ```
   include 'mpif.h'
   include 'dmumps\_struc.h'
   ```

   and ensure makefile specifies library file locations

2. Identify `MUMPS` package required and data structure name
   add line:

   ```
   TYPE (DMUMPS_STRUC) id
   ```

3. Name error variable `ierr`
   add line:

   ```
   INTEGER ierr
   ```

4. Define if problem is un-symmetric
   add line:

Variable	Description	Size	Stored on host or local?
NZ	Number of entries being inputed into the definition of matrix A	1	host
IRN	Integer array containing row indices for the matrix A entries	NZ	host
JCN	Integer array containing column indices for the matrix A entries	NZ	host
RHS	Real array containing values in the vector on the right hand side of the equation	N	host
NZ_loc	Number of entries stored locally on a processor being inputed into the definition of matrix A	1	local
IRN_loc	Integer array containing row indices for entries into matrix A calculated locally on processor	NZ_loc	local
JCN_loc	Integer array containing column indices for entries into matrix A calculated locally on processor	NZ_loc	local
A_loc	Real array containing matrix A entries calculated locally on processor, A_loc(k) corresponding to the indices of IRN_loc(k) and CN_loc(k)	NZ_loc	local

Table 3.2: List of `MUMPS` variables for distributed assembled problems

```
id%SYMM=0
```

5. Ask host core to be involved in factorisation
   add line:

   ```
   id%PAR=1
   ```

6. Initialise instance of `MUMPS` package
   add lines:

   ```
   id%JOB=-1
   CALL DMUMPS(id)
   ```

7. Allocate and assign `MUMPS` variables as listed in Table 3.2 using information already generated by `ParaFEM` such as array `storkm_pp` containing the stiffness matrix information associated with each core's assigned elements and array `r_pp` containing force vector information associated with each core's assigned elements.

8. call `MUMPS` package for solution
   add lines:

   ```
   id%JOB=6
   CALL DMUMPS(id)
   ```

9. Assemble solution on host
   add lines:

```
IF(numpe==1)THEN
  WRITE(*,*)'Solution is,', id%RHS
END IF
```

10. Destroy instance of MUMPS package
    add lines:

```
id%JOB=-2
CALL DMUMPS(id)
```

Chapter 4

Code development: extra features added to ParaFEM

The ParaFEM code has been extended by the author through adding the following features: (i) self weight, (ii) non-uniform pressure loading expressed in terms of the local element face co-ordinates, and (iii) error analysis. Throughout this chapter the theory behind the code features will be explained and an example problem shown to verify the code extension. Most of the example problems are based on a cube where one-eighth of the cube is modelled (taking advantage of three planes of symmetry). The cube is therefore subject to roller boundary conditions on its horizontal base and two vertical surfaces, see Figure 4.1. A pre-processor mesh refinement algorithm has also been developed. This will be presented alongside the error analysis capability added into ParaFEM by the author. This algorithm can only deal with whole domain refinement, the limitations of this will be discussed in Section 4.3.3 where the benefits of local mesh refinement are also described. The history of local refinement of hexahedral meshes will be summarised and difficulties associated with such refinement highlighted. Finally principal stress plots and peak stress envelopes used in this project are described.

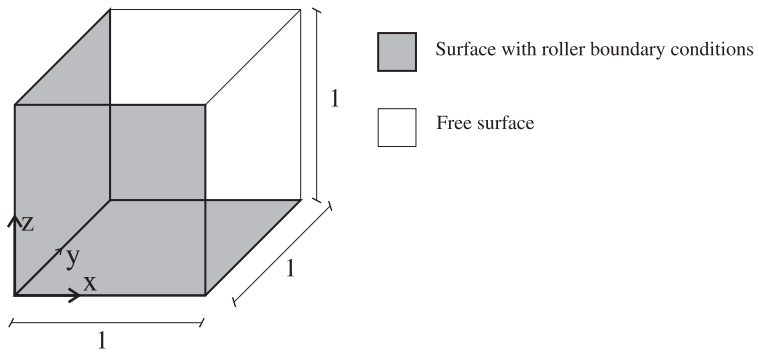


Figure 4.1: Boundary conditions for the unit cube used in the example problems

4.1 Self weight

All loadings on a finite element need to be modelled as nodal loads. These point loads, known as the equivalent nodal forces, can be calculated using the principle of virtual work. The infinitesimal gravitational force, dG, acting on a element of density, ρ, and volume, dV, can be written as

$$dG = -\rho g dV. \tag{4.1}$$

This gravitational force acts in the direction of the gravitational acceleration. If the direction of gravitational loading coincides with the global z axis then the components of, dG, are as follows,

$$\begin{aligned} dG_x &= 0 \\ dG_y &= 0 \\ dG_z &= -\rho g dV. \end{aligned} \tag{4.2}$$

The principle of virtual work states that a continuous body is in equilibrium if the virtual work of all the forces acting on the body is zero in a virtual displacement. Therefore the total work done by the gravitational force due to a virtual displacement must equal the sum of the work contribution from each node in a finite element in a virtual displacement. The total work done in a virtual displacement can be calculated by integrating the work done over the element using the element shape functions. This gives the expression

$$\{P_z\}w^* = \int_V [N]^T w^* \rho g dV, \tag{4.3}$$

where P_z is a vector of equivalent nodal forces for each node acting in the z direction such that for an eight noded element,

$$\{P_z\} = \begin{Bmatrix} P_{z_1} \\ P_{z_2} \\ P_{z_3} \\ P_{z_4} \\ P_{z_5} \\ P_{z_6} \\ P_{z_7} \\ P_{z_8} \end{Bmatrix}, \tag{4.4}$$

w^* is the virtual displacement applied to the element and $[N]$ is a matrix of shape function associated with the nodes such that for an eight noded element,

$$[N] = [N_1 N_2 N_3 N_4 N_5 N_6 N_7 N_8]\,. \tag{4.5}$$

Since the virtual displacement equation holds for all values of w^*, the integration term is independent of w^* and this term can therefore be cancelled from both sides of the equation. The integral expression for P_z can be calculated easily using Gaussian quadrature, as follows.

$$\{P_z\} = \sum_{i=1}^{nGp} [N]\rho g w \det(J), \tag{4.6}$$

where w is the weight function associated with each Gauss point i, $\det(J)$ is the determinant of the Jacobian matrix, $[N]$ is the matrix of the shape functions associated which each of the nodes at Gauss point i. and nGp is the number of Gauss point used in the integration scheme. This calculation must be done for each node in the element to find all the nodal equivalent forces (the shape function, N, will also depend on the node being considered).

4.1.1 Self weight example

To confirm that the self weight capability has been successfully added into `ParaFEM` a simple analysis was carried out on a unit cube (see Figure 4.1 for boundary conditions). No external forces are applied but it is assumed that the cube has a density of $2400kgm^{-3}$. If the acceleration due to gravity is taken as $g = 9.81ms^{-2}$ acting in the z direction, then the total vertical force acting on the unit cube is $\int_V \rho g dV$. This would give a total vertical load of $23.54kN$. FEA was carried out on four different meshes to verify the subroutine, see Figures 4.2(a)-(d). 8-noded hexahedral elements were used with 8 Gauss points per element. The results confirm that the force f_z due to gravity is being calculated correctly. The total loads

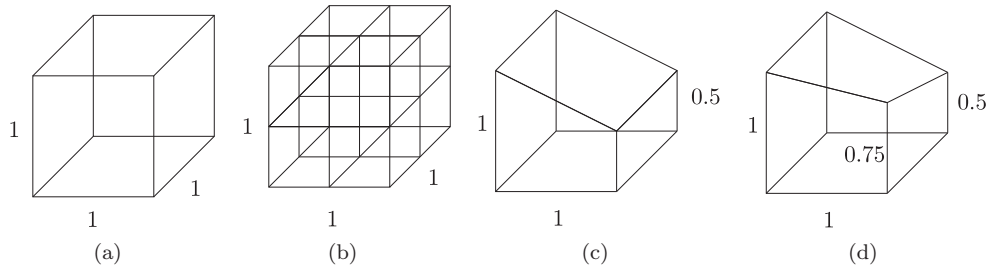


Figure 4.2: Self weight capability example problems

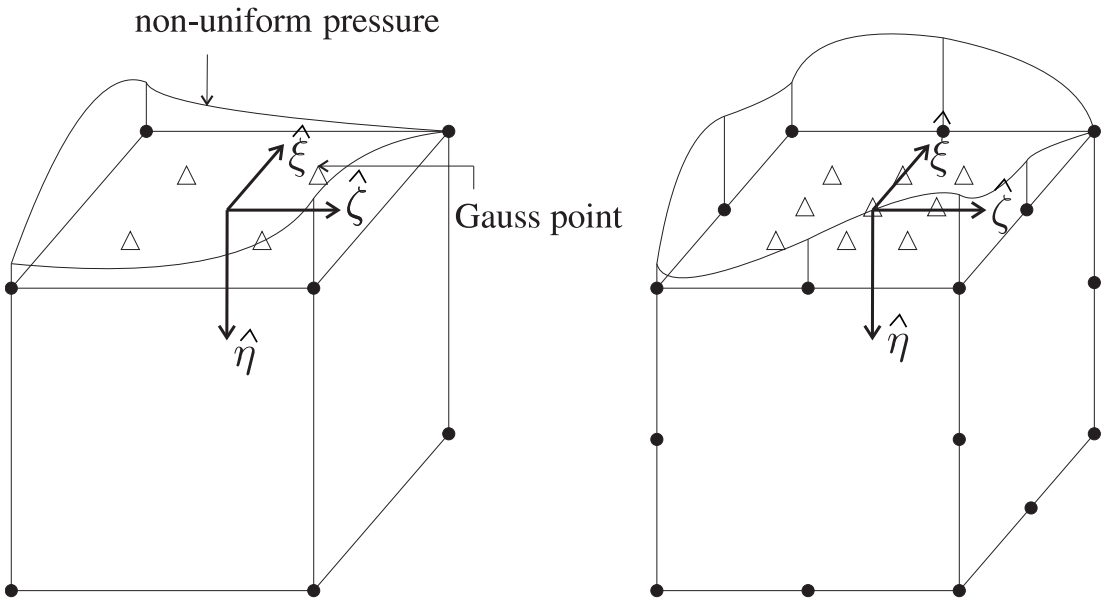
for each analysis are recorded in Table 4.1.1 .

Mesh	Volume (m^3)	$f_x(kN)$	$f_y(kN)$	$f_z(kN)$	FEA $f_z(kN)$
(a)	1.0000	0	0	23.54	23.54
(b)	1.0000	0	0	23.54	23.54
(c)	0.7500	0	0	17.66	17.66
(d)	0.6186	0	0	16.19	16.19

Table 4.1: Self weight example problem results (total forces)

4.2 Pressure loading

To calculate the nodal equivalent forces corresponding to non-uniform pressures acting on the element faces a similar Gaussian integration scheme was used where four or nine Gauss points were set up over the pressure surface (depending on whether 8 or 20-noded hexahedral elements were being considered), see Figure 4.3. The pressure was applied on an element face in a direction defined by a local co-ordinate. That is, either normal to the face ($\hat{\eta}$) or tangential to the face ($\hat{\xi}$ or $\hat{\zeta}$). There are three steps required to calculate the equivalent nodal forces from the local nodal pressures:



(a) 8-noded element (b) 20-noded element

Figure 4.3: Integration schemes for pressure loading

1. Step 1: calculate the local Gauss point pressures from the local nodal pressures.

2. Step 2: calculate the global Gauss point pressures from the local Gauss point pressures.

3. Step 3: calculate the global nodal forces from the global Gauss point pressures.

Firstly the local Gauss point pressures must be calculated (Step 1). This is possible by using the shape functions. The expression for the pressures in each of the local directions p_{Gp_ζ}, p_{Gp_η} and $p_{Gp_{\hat{\xi}}}$ are as follows

$$p_{Gp_{\hat{\zeta}}} = \sum_{i=1}^{nn} N_i p_{i_{\hat{\zeta}}} \qquad p_{Gp_{\hat{\eta}}} = \sum_{i=1}^{nn} N_i p_{i_{\hat{\eta}}} \qquad \text{and} \qquad p_{Gp_{\hat{\xi}}} = \sum_{i=1}^{nn} N_i p_{i_{\hat{\xi}}} \tag{4.7}$$

where i is the node number (i = 1...number of nodes (nn)) and $p_{i_{\hat{\zeta}}}$, $p_{i_{\hat{\eta}}}$ and $p_{i_{\hat{\xi}}}$ are the local nodal pressures associated which each of the nodes i. Secondly the local pressures are transformed into global pressures p_{Gp_x}, p_{Gp_y} and p_{Gp_z} at each of the Gauss points (Step 2). The expressions for global Gauss point pressure in the three global directions at each Gauss point are as follows

$$p_{Gp_x} = \frac{p_{Gp_{\hat{\zeta}}} \frac{\delta x}{\delta \hat{\zeta}}}{|\hat{\zeta}|} + \frac{p_{Gp_{\hat{\xi}}} \frac{\delta x}{\delta \hat{\xi}}}{|\hat{\xi}|} + \frac{p_{Gp_{\hat{\eta}}} varea_x}{det(J)} \tag{4.8}$$

$$p_{Gp_y} = \frac{p_{Gp_{\hat{\zeta}}} \frac{\delta y}{\delta \hat{\zeta}}}{|\hat{\zeta}|} + \frac{p_{G_{\hat{\xi}}} \frac{\delta y}{\delta \hat{\xi}}}{|\hat{\xi}|} + \frac{p_{G_{\hat{\eta}}} varea_y}{det(J)} \qquad \text{and} \tag{4.9}$$

$$p_{Gp_z} = \frac{p_{Gp_{\hat{\zeta}}} \frac{\delta z}{\delta \hat{\zeta}}}{|\hat{\zeta}|} + \frac{p_{G_{\hat{\xi}}} \frac{\delta z}{\delta \hat{\xi}}}{|\hat{\xi}|} + \frac{p_{G_{\hat{\eta}}} varea_z}{det(J)} \tag{4.10}$$

where

$$|\zeta| = \sqrt{((\frac{\delta x}{\delta \hat{\zeta}})^2 + (\frac{\delta y}{\delta \hat{\zeta}})^2 + (\frac{\delta z}{\delta \hat{\zeta}})^2)} \tag{4.11}$$

$$|\hat{\xi}| = \sqrt{((\frac{\delta x}{\delta \hat{\xi}})^2 + (\frac{\delta y}{\delta \hat{\xi}})^2 + (\frac{\delta z}{\delta \hat{\xi}})^2)} \tag{4.12}$$

$$varea_x = \frac{\delta z}{\delta \hat{\xi}}\frac{\delta y}{\delta \hat{\zeta}} - \frac{\delta y}{\delta \hat{\xi}}\frac{\delta z}{\delta \hat{\zeta}} \tag{4.13}$$

$$varea_y = \frac{\delta x}{\delta \hat{\xi}}\frac{\delta z}{\delta \hat{\zeta}} - \frac{\delta z}{\delta \hat{\xi}}\frac{\delta x}{\delta \hat{\zeta}} \tag{4.14}$$

$$varea_z = \frac{\delta y}{\delta \hat{\xi}}\frac{\delta x}{\delta \hat{\zeta}} - \frac{\delta x}{\delta \hat{\xi}}\frac{\delta y}{\delta \hat{\zeta}} \tag{4.15}$$

and

$$det(J) = \sqrt{(varea_x)^2 + (varea_y)^2 + (varea_z)^2}. \tag{4.16}$$

Finally once the global Gauss point pressures have been obtained we can use Gaussian quadrature to obtain an expression for the equivalent nodal forces. The expression for these nodal forces are as follows,

$$p_x = \sum_{i=1}^{nGp} N_i p_{Gp_{x_i}} w_i \det(J) \qquad p_y = \sum_{i=1}^{nGp} N_i p_{Gp_{y_i}} w_i \det(J) \qquad \text{and} \qquad p_z = \sum_{i=1}^{nGp} N_i p_{Gp_{z_i}} w_i \det(J) \tag{4.17}$$

where i is the Gauss point number, N_i is the shape function for the node under evaluation at Gauss point i and w_i is the weight function for Gauss point i. This calculation must be carried out for each node.

4.2.1 Pressure loading example

Uniform pressure applied to a unit cube

Note: the boundary conditions for this problem are described at the beginning of this chapter and can be viewed in Figure 4.1 When a uniform pressure of $1MPa$ is applied in the z direction to the top surface (perpendicular to the $x-y$ plane) of a unit cube containing 27 equally sized 8-noded elements, we can expect to see a uniform stress distribution of $1MPa$ at each of the Gauss points throughout the cube. To check that this is the case, the principal stress vectors at each integration point have been plotted (Figure 4.5). This plot shows that under these loading conditions only principal stresses in the z direction are generated, this is because the uniform pressure in the z direction generates nodal displacements in the z direction only.

Linearly varying pressure applied to a unit cube

When a linearly varying pressure (from $0MPa$ at $y=0$ to $1MPa$ at $y=1$) is applied in the z direction to the top surface (perpendicular to the $x-y$ plane) of a unit cube, one would expect a non-uniform variation in the stress distribution within the cube due to non-uniform nodal force vectors and nodal displacement vectors. Such deformation means that varying shear stresses are generated in the material,

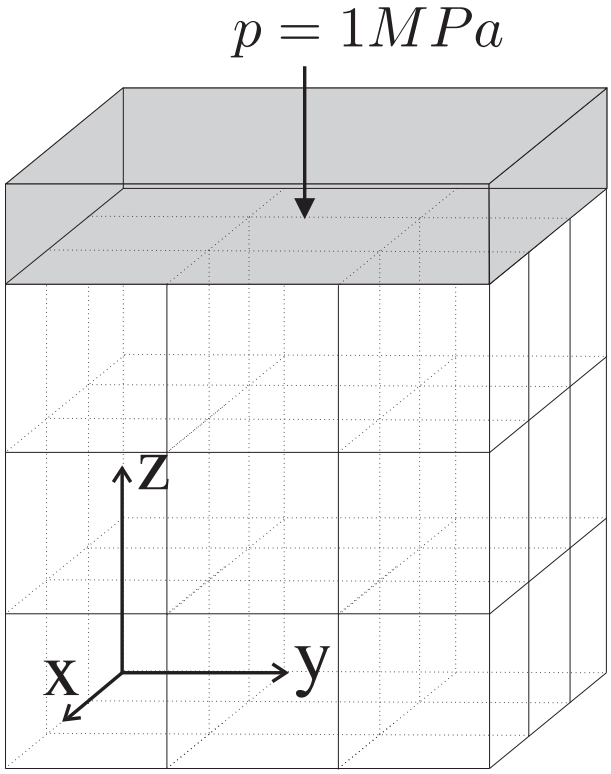


Figure 4.4: Uniform pressure load on the example cube

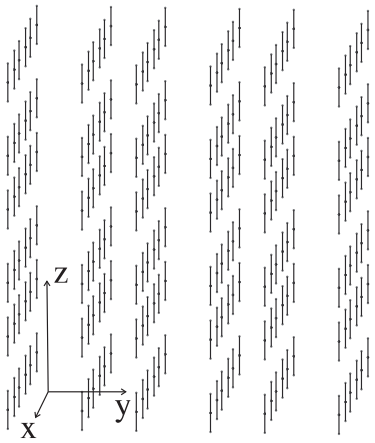


Figure 4.5: Gauss point principal stress vectors resulting from a uniform pressure load

this means that the principal stress directions also vary throughout the structure. To check that this is the case, the principal stress vectors have been plotted at each of the Gauss points (see Figure 4.7).

4.3 Mesh refinement

The accuracy of a FEA solution can be increased by using more elements in areas where the stress gradient varies in a manner not fully captured by a single element. This process is called mesh refinement. Mesh refinement will increase the computational requirements of the problem and therefore should only be

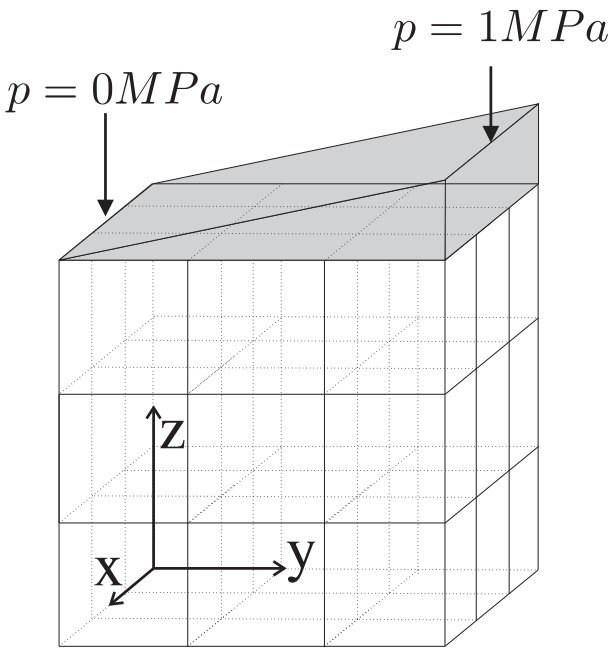


Figure 4.6: Linearly varying pressure acting on the example cube

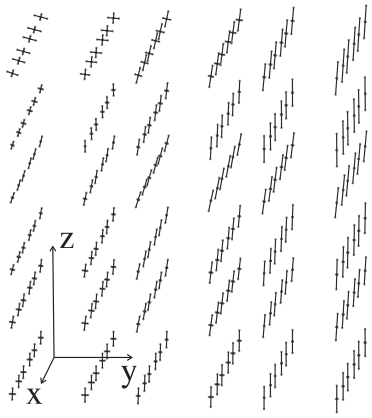


Figure 4.7: Gauss point stresses due to a linearly varying pressure load

carried out when necessary. To identify when mesh refinement is necessary an error measure should be used. This section focuses on mesh refinement and the following section describes error analysis, however the two are closely linked and rely upon each other when implemented within a FEA code.

The two major classes of mesh refinement are h-refinement and p-refinement. h-refinement describes a form of refinement where the element type is kept the same, but the number of elements is changed. Alternatively, p-refinement is a method whereby the order (type) of the element is increased whilst keeping the number of elements unchanged [26]. The Fortran code `refine_mesh.f90` was written by the author to carry out a remeshing scheme where a process of progressive h and p-refinement is applied to hexahedral elements. This code requires a `ParaFEM` co-ordinate and 8-noded element topology input file called `p121.d`, see Table 3.1. Firstly the order of each element is increased from eight nodes to twenty nodes, then each 20-noded element is split into eight 8-noded elements. The process can be repeated, see Figure 4.8. For

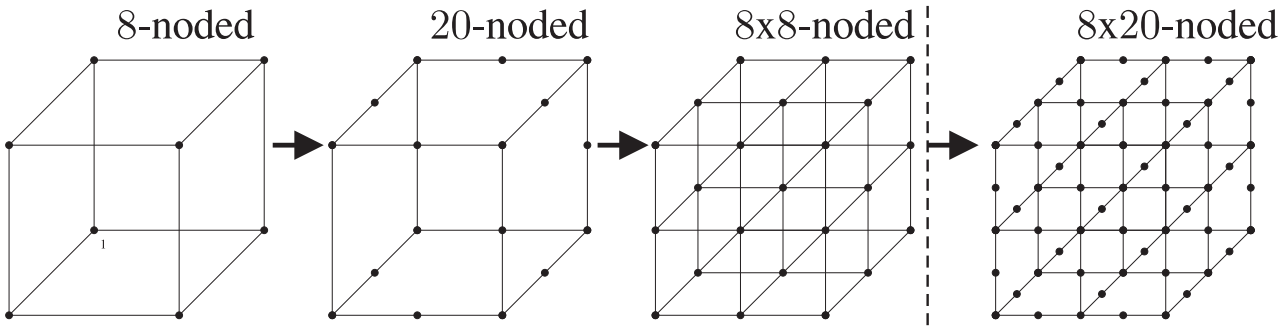
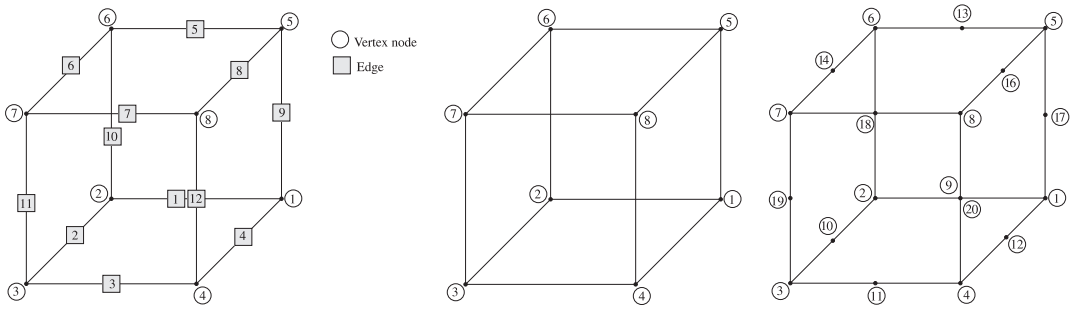


Figure 4.8: Mesh refining process

each conversion step, 8-noded mesh → 20-noded mesh and 20-noded mesh → 8 × 8-noded mesh, new nodal coordinates and element topologies must be determined. The next two subsections describe how these conversions are carried out within the Fortran code. To distinguish between mesh arrays, array names are followed by an 8 for the 8-noded mesh, 20 for a 20-noded mesh and 88 when the 20-noded elements are split into 8 × 8-noded elements.

4.3.1 *p*-refinement: 8 → 20-noded elements



(a) Edge and node numbering for mesh refinement 8 → 20

(b) 8-noded element `etpl`

(c) 20-noded element `etpl`

Figure 4.9: `etpl` node ordering

The following 5 steps describe the process of *p*-refinement for the hexahedral elements:

1. For each element, determine the two vertex nodes associated with each edge
2. For each element, loop around each edge $1-8$ and check if the mid-edge node associated with the edge has been assigned a node number by checking vertex nodes against the values stored in the new-node array `newn(total number of new nodes, Corner Node Values 1:2)`. If the vertex node associated with the edge of the element is not already stored in `newn`, then store the new-node number and associated vertex nodes in `newn` array.
3. For each mid-edge node on each edge assign the correct node number to the element topology array (`etpl20`). Note the element topology for the first eight nodes of the 20-noded hexahedral correspond to the 8-noded hexahedral element topology `etpl8`, see Figure 4.9(b) . The additional twelve terms in `etpl20` correspond to the mid-edge nodes, see Figure 4.9(c).

Edge, see Figure 4.9(a)	Vertex node 1	Vertex node 2
1	1	2
2	2	3
3	3	4
4	4	1
5	5	6
6	6	7
7	7	8
8	8	5
9	1	5
10	2	6
11	3	7
12	4	8

Table 4.2: Table to show the verteces associated with each new edge node for *p*-refinement

Surfaces see Figure 4.10	Vertex Node 1	Vertex Node 2	Vertex Node 3	Vertex Node 4
1	3	7	8	4
2	4	8	5	1
3	1	5	6	2
4	2	6	7	3
5	5	6	5	8
6	2	3	4	1

Table 4.3: Table to show the vertex nodes associated with each new surface node for *h*-refinement

4. For each new node calculate its coordinates and store in the `coord20` array. These coordinates are given by half the sum of the vertex node `cn` co-ordinate values in each global co-ordinate direction $1-3$, for example, `coord20(newn,1)=(coord8(cn1,1)+coord8(cn2,1))/2` .

5. Once `etpl20` and the new node coordinates have been established, the new input files can be written.

4.3.2 *h*-refinement: each 20-noded element → eight 8-noded elements

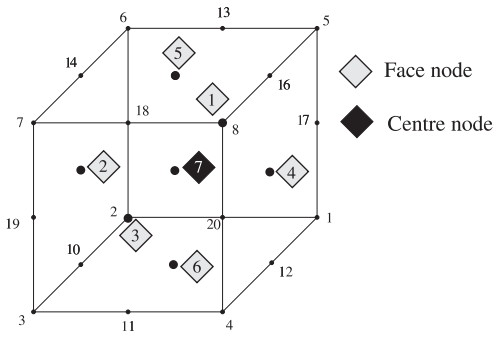


Figure 4.10: New nodes for 8 x 8-noded mesh

The following 6 steps describe the process of *h*-refinement for the hexahedral elements:

1. For each element, determine the 4 vertex nodes associated with each new face node and the eight

vertex nodes used to define the new centre node (Figure 4.10).

2. For each element, loop around each face $1-6$ and check if the face-node associated with the four vertex nodes has been assigned a new node number by checking the vertex nodes against the values stored in the array `newn(total number of new nodes, vertex node values 1:4)`. If a new node is found, give that node a new node number and store the vertex nodes in he `newn` array.

3. For each face node, assign the correct node number to the array `elnewn(1:7)`.

4. For each central node, assign a new node number to the array `elnewn(1:7)`. Note, there will be seven new nodes associated with each original 20-noded element.

5. For each 20-noded element, loop through the eight new sub-elements and assign `etpl88` from `etpl20` and `elnewn`. The nodes associated with each subelement (a) to (h) are shown in Figure 4.11.

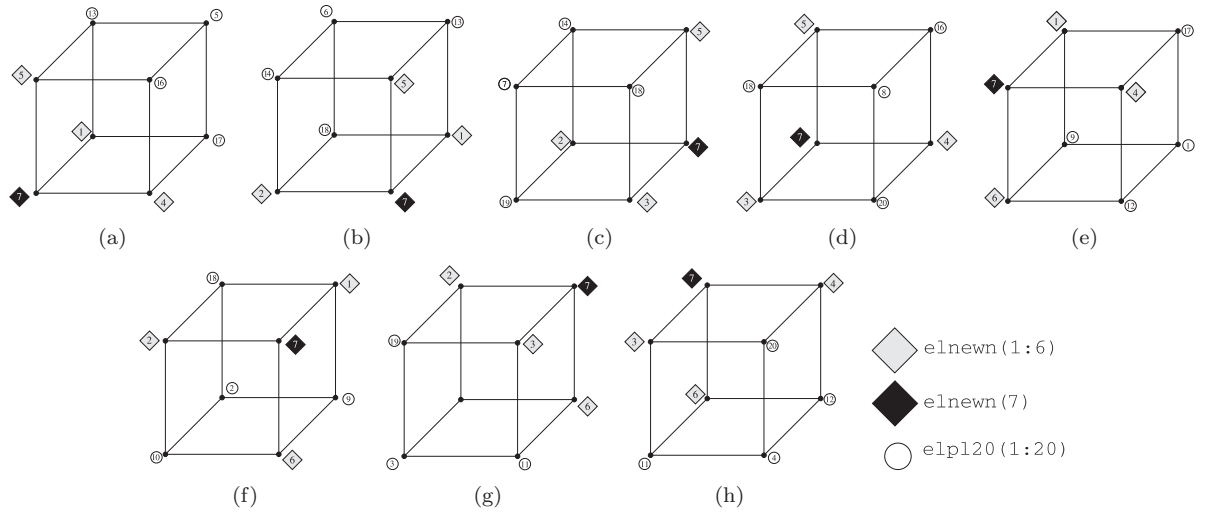


Figure 4.11: Sub-elements 1-8 node numbering scheme

6. For each new surface node, calculate its coordinates and store in the `coord` array. The coordinates are given by one quarter of the sum of the vertex node `cn` co-ordinate values in each global co-ordinate direction $1-3$, for example, `coord88(newn,1)=(coord20(cn1,1)+coord20(cn2,1) +coord20(cn3,1)+coord20(cn4,1))/4`.

7. For each new central node calculate its coordinates and store in the `coord` array - an eigth of the sum of the vertex node `cn` co-ordinate values in each global co-ordinate direction $1-3$, for example, `coord88(newn,1)=`$\frac{1}{8}$`*(coord20(cn1,1)+coord20(cn2,1) +coord20(cn3,1)+coord20(cn4,1)+ coord20(cn5,1)+coord20(cn6,1) +coord20(cn7,1)+coord20(cn8,1))`.

4.3.3 Local mesh refinement

Unacceptable errors in the stress field may appear in a local region within the whole domain. Use of a re-meshing algorithm which refines the whole domain can be wasteful, and it is therefore necessary to consider local remeshing strategies. Such strategies require a transition zone between the original coarse meshed areas and the new fine meshed areas. Difficulty arises when generating all hexahedral meshes in

a conforming way, that means when elements join to form a mesh they connect together in such a way that no nodes are left intersecting with an element and not being used to define its topology (such a node is called a hanging node). This is relatively simple when considering triangular and tetrahedral meshes, however for hexahedral elements the procedure is more complicated. Note that none of the schemes described in this section (4.3.3) were implemented in `ParaFEM` due to time constraints. This section had been included in order to point the way forward for future developments.

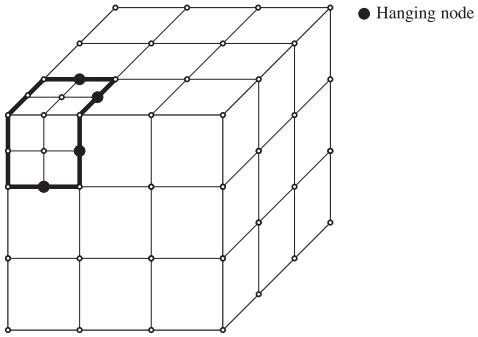
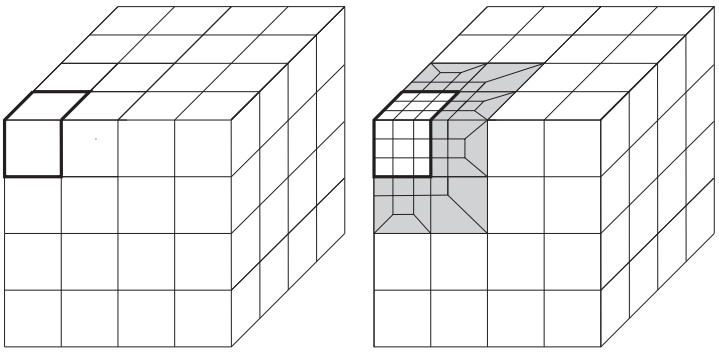


Figure 4.12: Unconforming hexahedral 8-noded mesh

Two different options are available for hexahedral 8-noded element refinement: two-refinement (2-Ref) and three-refinement (3-Ref) schemes. 2-Ref divides each hexahedral element into eight new elements whilst 3-Ref divides each hexahedral element into twenty-seven new elements. Both these elements form a transition zone by joining template elements together. Template elements model the possible ways an element could be sub-divided whilst maintaining conformity. A family of template elements can be produced which represent all possible ways of subdividing an element for a given refinement scheme. These template elements usually model a sub-division pattern which allows the element to match a coarse mesh on one face and a more refined mesh on another face allowing connection. Example of the 2-Ref and 3-Ref templates are given in Figures 4.16 and 4.17 respectively. In many cases 2-Ref will produce a mesh with fewer elements and a smoother transition zone. However, complications arise when developing a robust 2-Ref algorithm, because pairs of adjacent elements must be considered. This is not the case for 3-Ref where refinement can be carried out locally. This means that the transition templates fit together irrespective of the geometry of the refined area as the templates do not rely on pairs of elements to generate a conforming zone. These templates are shown in Figure 4.13(b). Template-based refinement methods have been widely used for 8-noded hexahedral mesh generation [11]. Most of these methods use 3-Ref templates, originally introduced by Schneiders in 1996 [21]. He proposed four 3-Ref templates for node, edge, face and volume refinement (see Figure 4.16). The elements in the area of refinement are subdivided using the volume template (d), then neighbouring elements are subdivided using one of the four templates, (a), (b), (c) or (d) to generate a transition zone. Other templates (one is shown in Figure 4.14) are also available but they do not provide a stable refinement. The latter refers to the condition where the minimum element internal angle of the refined mesh does not depend on the subdivision level [21]. Figures 4.14(a) and (b) show how the quality of the elements reduces as this template is used to refine the mesh. When the elements get very elongated as shown in 4.14(b), they will no longer provide a good estimation for the stress distribution.

2-Ref is limited to structured meshes (meshes which conform to specific geometric characteristics). The



(a) Unrefined mesh

(b) 3-Ref refined mesh with highlighted transition zone

Figure 4.13: 3-Ref example

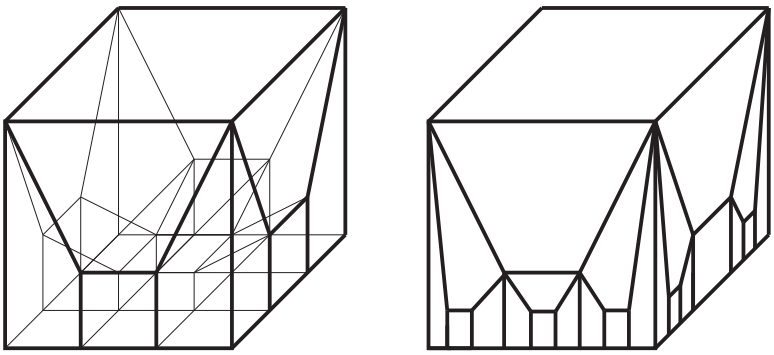
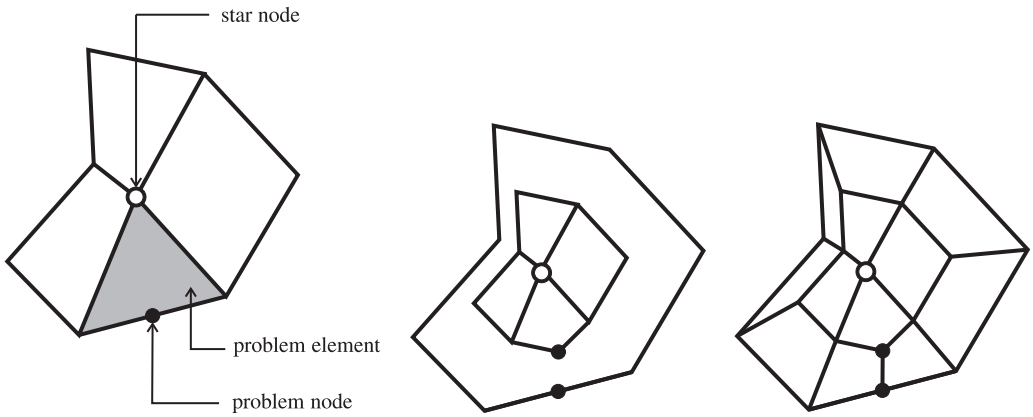


Figure 4.14: 3-Ref unstable template

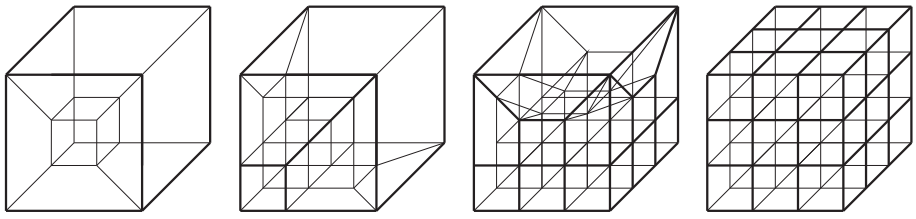
use of such strategies can result in large volumes of the mesh being replaced by template (d). They can only be used robustly on convex domains. Another method of refinement (also suggested by Schneiders) is a shrink-and-connect strategy based on *pillowing* [17]. Pillowing is the name given to this technique which essentially identifies a *problem node* and one of its *problem elements*. A group of elements is then defined by taking a different node (the *star node*) from the *problem element* and forming a group of elements all of which share the *star node* (this group is called the *shrink set*). The shrink set is then shrunk, the *problem element* re-shaped and finally the whole group is connected in a conforming way to the surrounding elements. Figure 4.15 shows the basic concept of a *pillowing* strategy for 2D quadrilateral elements. This method extends Schneider's original template based method by providing additional templates generated by *pillowing* element layers in alternating x, y, and z directions. Additional work has been provided by Tchon *et al.*. [23], Marechal [15], Yamakawa *et al* [24], Harris [9], Zhang *et al.* [25], Parrish *et al.* [25] and Ito *et al.* [11] mostly focusing on providing more robust 3-Ref strategies. Ebeida *et al.* [7] have implemented Schneiders 2-Ref template, see Figure 4.17, using two novel methods overcoming concavity restrictions, and providing an algorithm which works for unstructured meshes and parallel implementation. Method 1 simply implements 2-Ref templates in a new way suitable for parallel implementation but problems with generating a conforming mesh and fitting templates together still exist. Method 2 adds an additional *pillowing* capability, overcoming a restriction of Method 1 involving template connectivity and the interaction of transition zones. The last of these restrictions occurs when the distance between refined zones is only one element thick. The orientation of the template inserted due



(a) Find a shrink set (b) Shrink (c) Connect

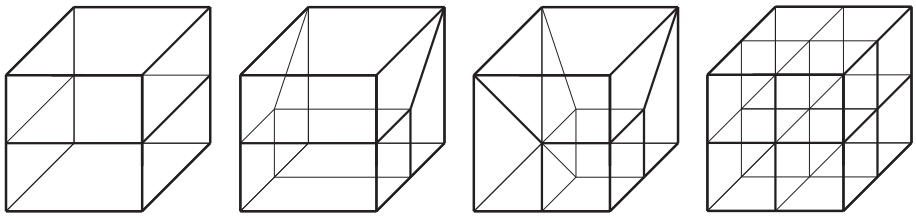
Figure 4.15: *Pillowing* of a quadrilateral mesh [7]

to one refined zone will conflict with the required orientation of the template inserted due to the other refined zone. In this case one template orientation must be chosen. This results in hanging nodes on the boundary between the transition zone and the other refined zones. Method 2 uses a *pillowing* technique to resolve this problem. This method only requires template A, see Figure 4.17(a), then subsequent hanging nodes are dealt with by a 3D pillowing technique.



(a) (b) (c) (d)

Figure 4.16: Four 2-Ref refinement templates by Schneiders *et al* for (a) node; (b) edge; (c) face; (d) volume



(a) (b) (c) (d)

Figure 4.17: Three 3-Ref refinement templates by Schneiders *et al* for (a) A; (b) B; (c) C and (d) D

4.4 Error analysis

Mesh refinement will increase the computational effort of the FEA and therefore will normally only be carried out when the jumps in the local stress fields between adjacent elements are unacceptable. An error

indicator is required to identify when this is the case. To analyse the errors associated with the FEA solution, the Zienkiewicz and Zho [27] error estimator (referred to here as the Z^2 error estimator) has been implemented into `ParaFEM` for the 8-noded and 20-noded 3D hexahedral elements. This estimator uses a Super-Convergent Patch Recovery (SPR) technique [28] to estimate the nodal stresses according to a polynomial expansion. These estimated stresses are called recovered stresses $\{\sigma^*\}$ and are used instead of the exact stresses, $\{\sigma\}$, to calculated the error in stress e_σ.

The stress error can be defined as the difference between the approximate FEA solution, $\{\sigma_h\}$ and $\{\sigma\}$. For nodal stresses this can be written as

$$\{e_\sigma\} = \{\sigma\} - \{\sigma_h\}\,. \tag{4.18}$$

However, the exact stress is not known, therefore $\{\sigma^*\}$ is substituted into (4.18) to obtain an estimated error meassure

$$\{e_\sigma\} \approx \{e_\sigma^*\} = \{\sigma^*\} - \{\sigma_h\}\,. \tag{4.19}$$

The energy norm of the error $\|e\|$ is used as an appropriate measure to compare errors within domains. This is written as

$$\|e\| = \left(\int_\Omega \{e_\sigma^*\}^T [D]^{-1} \{e_\sigma^*\}\, d\Omega\right)^{\frac{1}{2}}. \tag{4.20}$$

Integration is carried out using numerical Gaussian quadrature summing the contribution from each Gauss point ($i = 1...nGp$) in each element using the following expression,

$$\|e\| = \left(\sum_{i=1}^{nGP} \{e_\sigma^*\}^T [D]^{-1} \{e_\sigma^*\}\, \omega_i \;\; det(J_i)\right)^{\frac{1}{2}}. \tag{4.21}$$

In the following subsection, the SPR technique will be described and then a benchmark problem will be presented to verify this recovery process for both 8 and 20-noded hexahedral elements.

4.4.1 Calculating recovered nodal stresses within a 3D mesh using SPR

The SPR technique is used for recovering nodal stresses. This method assumes that each stress component in $\{\sigma^*\}$ belongs to a polynomial expansion of the same order as the FEA shape functions used in the original FEA analysis. The polynomial expansion for each component of $\{\sigma^*\}$ is valid over a small discrete area of the mesh called a patch. In the next two subsections patches for 3D hexahedral meshes will be described. Firstly the general case where patches are within the domain are addressed and secondly special consideration is given to patches on the domain boundaries.

Setting up patches within the domain

A patch is defined by a group of elements surrounding a vertex node (otherwise known as the patch-node) where the approximation for $\{\sigma^*\}$ is calculated. This corresponds to eight nodes per element for both 8 and 20-noded hexahedral elements. Stresses at Gauss points within these patches are used to estimate $\{\sigma^*\}$ for the patch-node and other nodes lying within the boundary of the patch. Figure 4.18 shows an example patch for 8 and 20-noded hexahedral elements and identifies the patch-node and other nodes where $\{\sigma^*\}$ is calculated. For 8-noded hexahedral element patches within the domain, $\{\sigma^*\}$, is only estimated at the

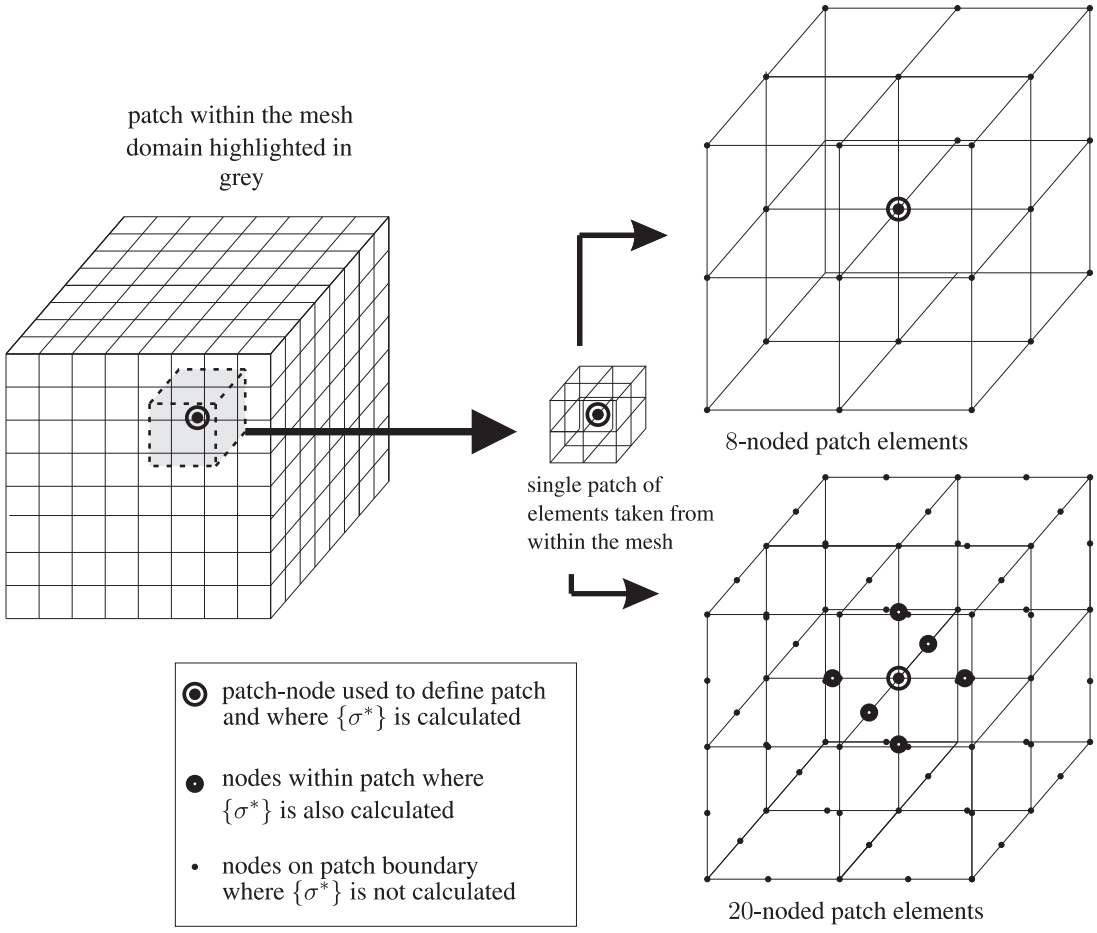


Figure 4.18: Setting up 8 and 20-noded patches within the domain

patch-node, but for 20-noded hexahedral elements additional stress vectors are estimated at the mid-side nodes within the element. In the example in Figure 4.18 a patch of eight elements is shown and $\{\sigma^*\}$ is estimated at an additional six mid-side nodes. When patch-nodes are not on a domain boundary the patch includes all elements surrounding the patch-node. However for boundary nodes the patches require special consideration. A description of how to choose the patch at domain boundaries is given in the following sub-section.

Setting up patches at domain boundaries

The general procedure for determining the patch elements has been described above. That process has to be adapted for patch-nodes at boundaries. The Z^2 paper [28] only describes SPR for 2D 9-noded quadrilateral elements only. In 2D quadratic meshes the general procedure presents a problem when patch-nodes are located at domain vertices. In this case patches consist of a single element and therefore provide insufficient stress data to determine $\{\sigma^*\}$. This is because a single Gauss point stress vector cannot be used to determine a 2D linear stress field through a 4-noded quadrilateral element and four Gauss point stress vectors cannot be used to determine a 2D quadratic stress field through an 8-noded or 9-noded quadrilateral element. The patch has to be extended so that an interior patch of four elements is used to determine $\{\sigma^*\}$ at the patch-node and other nodes within the element at the domain boundaries

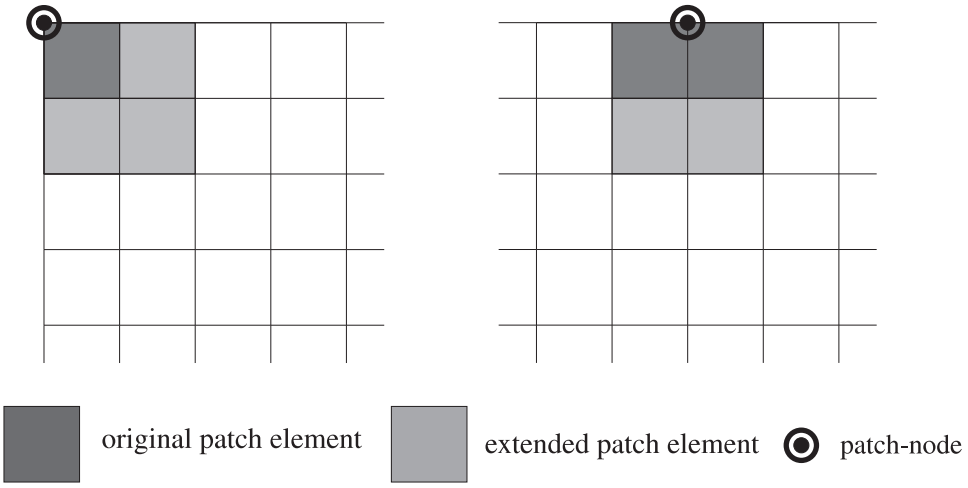
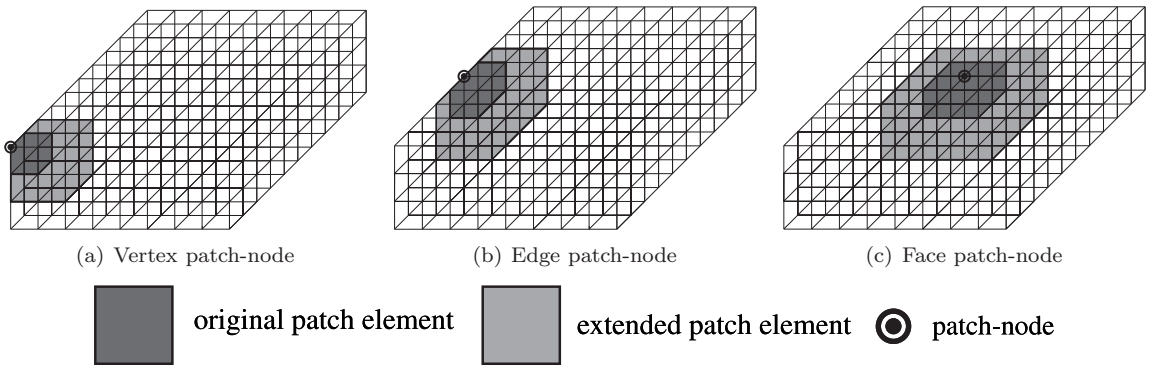


Figure 4.19: 2D quadrilateral element patch extensions at domain boundaries

(see Figure 4.19).

For 3D meshes there are three boundary cases which require special consideration, vertex patch-nodes, face patch-nodes and edge patch-nodes all generate inadequate patches in order to estimate $\{\sigma^*\}$. To deal with this problem the author has devised a rule to determine the patches in such situations. The rule states that if a patch-node is located on a domain boundary (at a vertex, on a surface or at an edge) the original patch must be extended. The extended patch will include all elements sharing nodes with elements already part of the original patch. Figures 4.20 shows how in these situations the patch is extended to include the additional elements resulting in a usable patch. In some cases the size of the patch increases by a factor of eight. This may seem an unnecessary expansion (since fewer elements could be used to determine a stress field), however the author's approach ensures there is no directional bias introduced into the recovered stress estimate due to the shape of the patches.



(a) Vertex patch-node (b) Edge patch-node (c) Face patch-node

Figure 4.20: 3D hexahedral element patch extensions at domain boundaries

Recovering stesses through each patch

$\{\sigma^*\}$ is estimated by considering each component of the stress vector in turn, the polynomial expansion for these stress components can be split into its two components; $\{\hat{p}\}$ containing the polynomial terms

and $\{a\}$ containing the polynomial coefficients such that

$$\{\sigma^*\} = \{\hat{p}\}^T \{a\}\,. \tag{4.22}$$

For 8-noded hexahedral elements the polynomial expansions will be linear and contain x, y and z components such that,

$$\{\hat{p}\} = \{1, x, y, z\} \tag{4.23}$$

and

$$\{a\} = \{a_1, a_2, a_3, a_4\}^T\,. \tag{4.24}$$

For 20-noded hexahedral elements the polynomial expansions will be quadratic and contain x, y and z components as follows,

$$\{\hat{p}\} = \left\{1, x, y, z, xy, yz, zx, x^2, y^2, z^2\right\} \tag{4.25}$$

and

$$\{a\} = \{a_1, a_2, a_3, a_4, a_5, a_6, a_7, a_8, a_9, a_{10}\}^T\,. \tag{4.26}$$

$\{a\}$ can be determined by assuming a least squares fit between the polynomial expansion and the FEA solution obtained at the Gauss points ($i = 1$...number of Gauss points per patch ($nGppP$))

$$F(a) = \sum_{i=1}^{nGppP} \left(\sigma_{h_i} - \sigma^*_{\hat{p}_i}\right)^2 \tag{4.27}$$

$$= \sum_{i=1}^{nGppP} \left(\sigma_{h_i} - \{\hat{p}_i\}\{a_i\}\right)^2 \tag{4.28}$$

By differentiating this expression with respect to a and setting it equal to zero we find

$$\sum_{i=1}^{nGppP} \{\hat{p}_i\}^T \{\hat{p}_i\}\{a_i\} = \sum_{i=1}^{nGppP} \{\hat{p}_i\}^T \{\sigma_{h_i}\} \tag{4.29}$$

This expression can be written and solved in matrix form

$$\{a\} = [A]^{-1}\{b\}\,. \tag{4.30}$$

We can see that

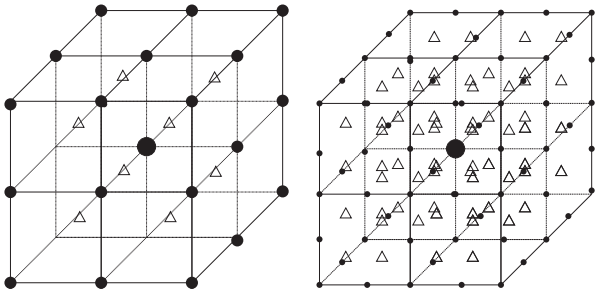
$$[A] = \sum_{i=1}^{nGPpP} \{\hat{p}_i\}^T \{\hat{p}_i\} \tag{4.31}$$

and

$$\{b\} = \sum_{i=1}^{nGPpP} \{\hat{p}_i\}^T \sigma_{h_i}\,. \tag{4.32}$$

For regular hexahedral meshes (where each interior node is surrounded by eight elements), each interior patch will consist of 8 elements. A reduced integration scheme will be used, such that $nGP = 1$ for 8-noded elements and $nGP = 8$ for 20-noded elements, as suggested by Zienkiewicz and Zhu [28]. This means that for our example the polynomial expansion will be fitted to a total of eight Gauss point values (nGppP=8) for the 8-noded elements and 64 Gauss point values (nGppP=64) for the 20-noded elements,

see Figure 4.21. Once both $[A]$ and $\{b\}$ have been calculated a linear solver must be used to obtain $\{a\}$.



(a) nGppP for the 8-noded hexahedral patch

(b) nGppP for the 20-noded hexahedral patch

Figure 4.21: SPR Gauss point integration schemes for hexahedral patches

We have used the `MUMPS` solver [18]. For each patch-node $[A]$ will be calculated once while $\{b\}$ and $\{a\}$ will be calculated six times, once for each stress component of $\{\sigma_p\}$. When a 20-noded hexahedral mesh is used, the recovered stress will be calculated at the mid-side nodes by at least two patches. In this case the mean value from the two patches would be taken, as suggested by Zienkiewicz and Zhu [28].

4.4.2 SPR error analysis example

The original Z^2 paper [28] which describes the SPR procedure, presented an example problem which considered a portion of an infinite plate with a central circular hole subject to a uniform pressure applied in the x direction. Plane strain conditions were assumed with a Poisson's ratio of $\nu = 0.3$ and Young's modulus $E = 1000$ (units were not given). The boundary conditions were prescribed such that on edges AE and CD symmetry conditions were imposed and on edges BC and DC the plate was loaded with tractions (calculated using the analytical solution for the stresses in an infinite plate), see Figure 4.22. The analytical solutions for the stress are expressed as

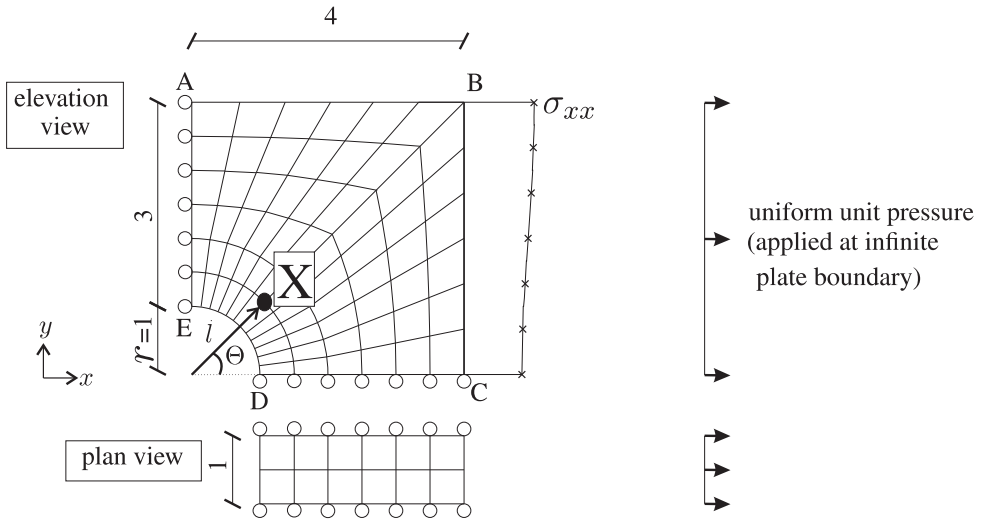


Figure 4.22: Infinite plate with a circular hole example problem

$$\sigma_x = 1 - \frac{l^2}{r^2}(\frac{3}{2}cos(2\Theta) + cos(4\Theta)) + \frac{3}{2}\frac{l^4}{r^4}cos(4\Theta),$$
$$\sigma_y = -\frac{l^2}{r^2}(\frac{1}{2}cos(2\Theta) - cos(4\Theta)) - \frac{3}{2}\frac{l^4}{r^4}cos(4\Theta) \qquad \text{and}$$
$$\sigma_{xy} = -\frac{l^2}{r^2}(\frac{1}{2}sin(2\Theta) + sin(4\Theta)) + \frac{3}{2}\frac{l^4}{r^4}sin(4\Theta). \tag{4.33}$$

The SPR algorithm implemented by the author in `ParaFEM` is specifically for 3D problems therefore this 2D problem was modeled in 3D. The mesh required 2 elements to model the thickness of the plate to enable the SPR procedure to capture the stress field in the z direction, see Figure 4.22.

Graphs a, b and c in Figure 4.23 illustrate how the recovered stresses at point X (with coordinates $1.0606602, 1.0606602, 0.500000$) converged to the analytical solution ($\sigma_{xx} = 1.1481481$, $\sigma_{yy} = -0.14814810$ and $\sigma_{xy} = -0.22222229$), as the mesh is refined. The error in energy norm per element has been calculated for each of these meshes. The results show how the error in the FEA reduces as the mesh is refined. Tables 4.4 and 4.5 show how both the maximum error (in energy norm per element) and the mean value over the whole domain reduces as the mesh is refined, for both the 8 and 20-noded elements.

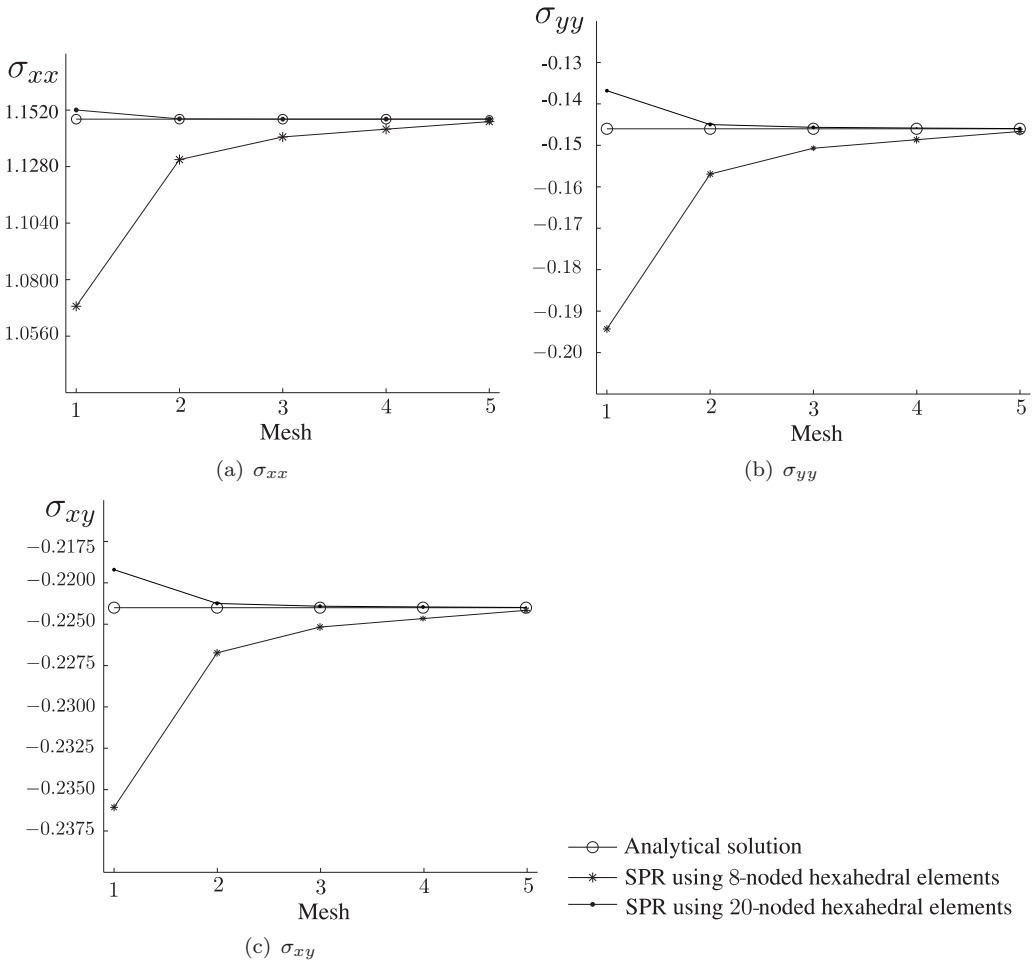


Figure 4.23: In-plane stresses at point X for meshes 1 to 5

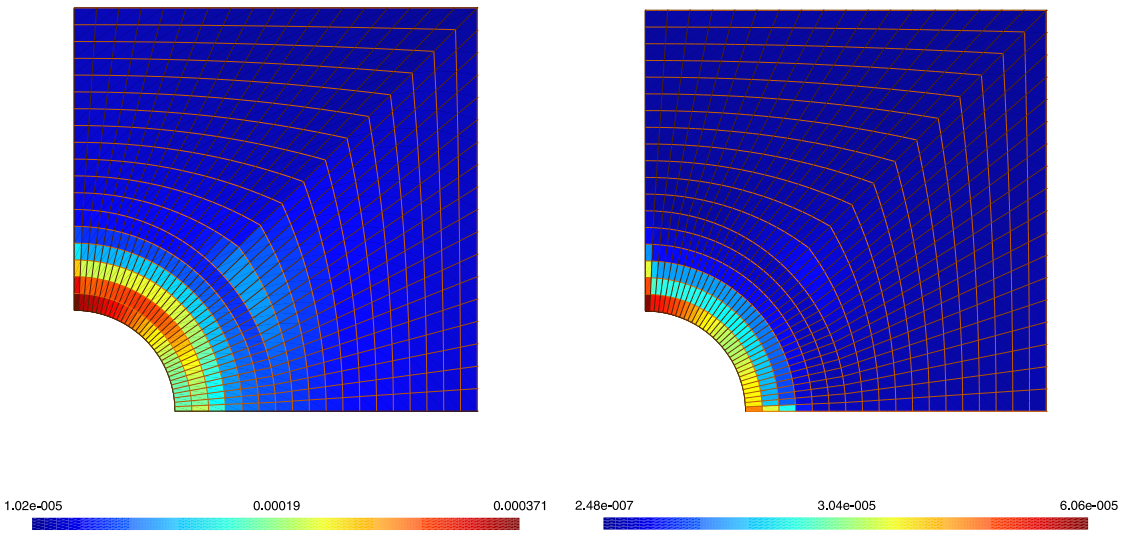
Mesh	No. of elements	$\overline{\lvert e \rvert}$	$\lvert e_{max} \rvert$
Mesh 1	144	0.6948×10^{-3}	2.0116×10^{-3}
Mesh 2	576	0.1784×10^{-3}	0.7241×10^{-3}
Mesh 3	1296	0.0776×10^{-3}	0.3706×10^{-3}
Mesh 4	2304	0.0426×10^{-3}	0.2261×10^{-3}
Mesh 5	9216	0.1003×10^{-3}	0.0657×10^{-3}

Table 4.4: SPR example 8-noded element results

Mesh	No. of elements	$\overline{\lVert \overline{e} \rVert}$	$\lVert \overline{e}_{max} \rVert$
Mesh 1	144	0.2123×10^{-3}	9.8158×10^{-3}
Mesh 2	576	0.2194×10^{-3}	0.1841×10^{-3}
Mesh 3	1296	0.0573×10^{-3}	0.6065×10^{-3}
Mesh 4	2304	0.0221×10^{-3}	0.2661×10^{-3}
Mesh 5	9216	0.0025×10^{-3}	0.0347×10^{-3}

Table 4.5: SPR example 20-noded element results

Figure 4.24 shows the result for the error analysis for Mesh 3 for both (a) the 8 and (b) the 20-noded elements. These figures show the error distribution throughout the structure and are closely comparable to plots included in [27]. The plots show high errors around the hole particularly around point E (see Figure 4.22).



(a) 8-noded elements (b) 20-noded elements

Figure 4.24: Energy norm error, $|e|$, visualisation for Mesh 3

4.5 Post processing

In this section post-processing methods used in this project will be described. Firstly three peak stress envelopes used to investigate structural integrity of concrete will be described. Secondly `Gmsh`, a piece of mesh generating software used in post-processing of the FEA results, will be introduced and finally principal stress plots will be described.

4.5.1 Peak stress envelopes

Peak stress envelopes are a way of describing the limiting stresses permissible when carrying out elastic analysis. A peak stress envelope can be visualised as a surface in principal stress space separating stress states which exceed the material's peak stress capacity and stress states which do not. When stress states lie outside the peak stress envelope during FEA, this suggests that cracking may be occurring in this part of the structure. This in itself does not indicate structural failure, however when large areas of the structure are experiencing stresses which lie outside the envelope, this could be highlighting a problem and indicate the structure has lost some (or all) of its integrity. Three different peak stress envelopes for concrete are used in this project to analyse the FEA results. Two multi-planar envelopes are used, a tension cut off envelope (Tenv) and a multi-planar envelope (MPenv). Figure 4.25 compares the Ottosen criterion and biaxial test data obtained by Kupfer *et al* [12] [13]. The plot shows that under bi-axial loading conditions (when the loading consists of a tensile and compressive component) the strength of concrete cannot be modeled accurately using a simple tension cut off envelope (an envelope where the peak stress equals the tensile strength of concrete in each principal direction). This is because the bi-axial experimental peak stress data recorded (where concrete is subject to compressive and tensile loads) is less than the tensile strength of concrete. MPenv has been developed by extrapolating from the biaxial data

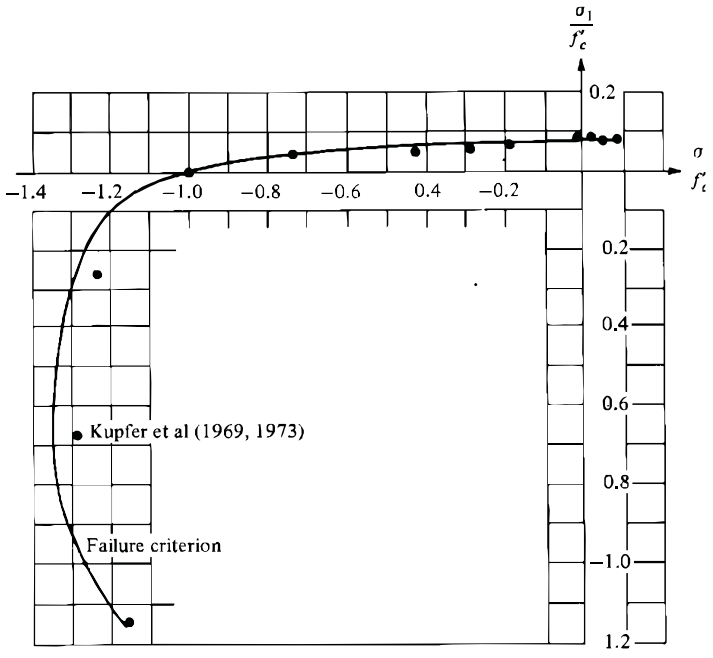


Figure 4.25: Bi-axial peak stress envelope showing tensile and compressive behaviour [5]

(this data is obtained more easily from experimental data). When the bi-axial envelope is extrapolated to the multi-axial case an envelope of this form is generated [4].

Planar peak stress envelope

A multi-planar peak stress envelope (MPenv) for concrete has been defined by considering nine intersecting planes. Three of the planes are described by the equation

$$a_1\sigma_1 + a_2\sigma_2 + a_3\sigma_3 - a_4 = 0, \tag{4.34}$$

where the constants a_1, a_2, a_3 and a_4 are given in Table 4.6. These planes generate the tension cut-off

a_1	a_2	a_3	a_4
1	0	0	f_t
0	1	0	f_t
0	0	1	f_t

Table 4.6: TCenv constants

envelope (TCenv). To determine whether the stress state lies inside the envelope, each component of the principal stress vector must be checked to determine whether σ_i is less than f_t. If this is the case, then the stress state is within TCenv. In order to capture the behaviour of concrete in the regions where the principal stress vector comprises of compressive and tensile components, six additional planes are required, described by

$$b_1\sigma_1 + b_2\sigma_2 + b_3\sigma_3 - b_4 = 0 \tag{4.35}$$

where the constants b_1, b_2, b_3 and b_4 are given in Table 4.7 and $f_c = -60MPa$ and $f_t = 3MPa$. The

b_1	b_2	b_3	b_4
$\frac{f_c}{f_t}$	1	0	f_t
1	$\frac{f_c}{f_t}$	0	f_t
0	$\frac{f_c}{f_t}$	1	f_t
0	1	$\frac{f_c}{f_t}$	f_t
$\frac{f_c}{f_t}$	0	1	f_t
1	0	$\frac{f_c}{f_t}$	f_t

Table 4.7: Additional constants for MPenv

envelope is shown in Figure 4.26. An algorithm has been developed to identify whether stress states are inside or outside the envelope. For each plane a stress state is declared to be inside or outside the envelope (if it is declared to be inside this does not mean it is fully inside the envelope but rather the criteria have been met for one out of the six planes). To determine which side of a plane a stress point is on, firstly a normal vector $\{n\}$ to the plane must be defined. The normal to a plane $b_1\sigma_1 + b_2\sigma_2 + b_3\sigma_3 - b_4 = 0$ is

$$\{n\} = \begin{Bmatrix} b_1 \\ b_2 \\ b_3 \end{Bmatrix},$$

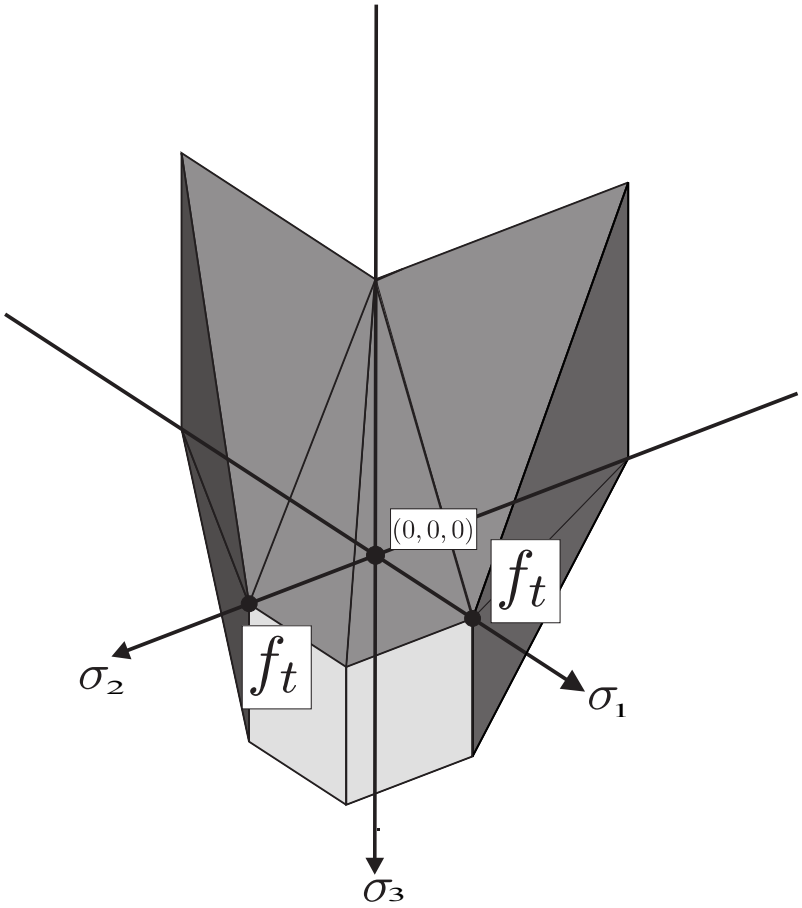


Figure 4.26: Concrete peak stress envelope

with the unit normal vector given by,

$$\{\hat{n}\} = \frac{1}{|n|} \begin{Bmatrix} b_1 \\ b_2 \\ b_3 \end{Bmatrix}.$$

An alternative way of expressing the equation for the plane $b_1\sigma_1+b_2\sigma_2+b_3\sigma_3-b_4 = 0$ is $\{n\}^T \{\sigma\}-b_4 = 0$. For the six planes considered the normal vectors are all directed outwards from the envelope, away from the origin of stress space. The shortest distance, l, of each stress state to the plane (the distance from the stress state along a line perpendicular to the plane) can then be calculated using a single stress state on the face $\{\hat{\sigma}\}$, see Figure 4.27. The equation to calculate l is,

$$l = \{\hat{n}\}^T \begin{Bmatrix} \sigma_1 - \hat{\sigma_1} \\ \sigma_2 - \hat{\sigma_2} \\ \sigma_3 - \hat{\sigma_3} \end{Bmatrix}. \tag{4.36}$$

Using l, a test stress state can be calculated $\{\sigma_{tss}\} = \{\sigma\} + |l|\{\hat{n}\}$. If this stress state $\{\sigma_{tss}\}$ lies on the plane ($\{n\}^T \{\sigma_{tss}\} - b_4 = 0$) the stress state has met the inside criteria for this plane. If the stress state lies within the envelope (when considering all six planes), then this inside criteria will be met for all six planes, and the stress state will also fall inside TCenv.

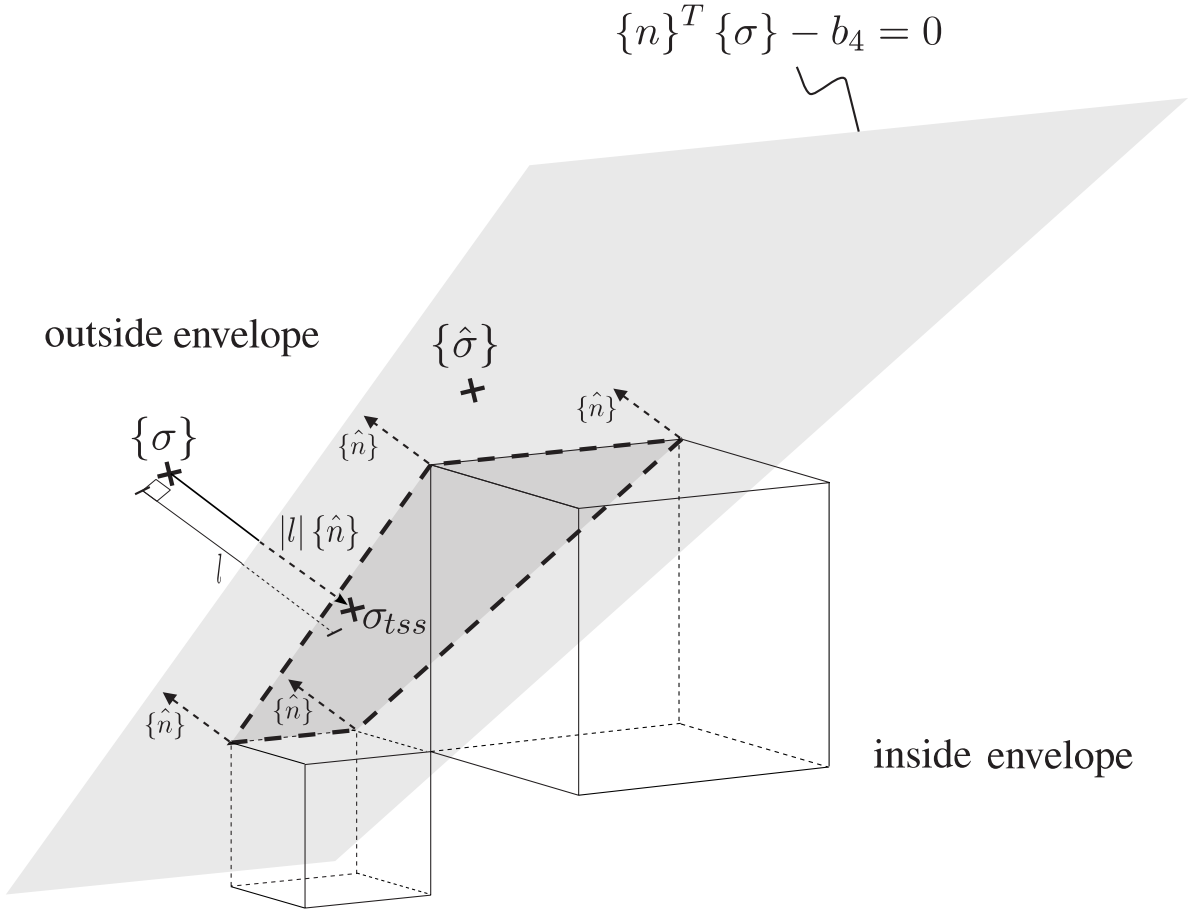


Figure 4.27: Figure to show how to determine whether stress state is inside or outside the peak stress envelope

4.5.2 A smooth peak stress envelope for structural concrete

This smooth peak stress envelope (Senv) was first presented in [14], and was initially developed as part of an advanced elasto-plasticity model to study the response of pre-stressed concrete nuclear reactor vessels. Senv is expressed in terms of (i) a deviatoric shape function, $\hat{r}$, (ii) the compression meridian, ϱ_{ps_c}, and (iii) the extension meridian, ϱ_{ps_e}. Both ϱ_{ps_c} and ϱ_{ps_e} are functions of the mean stress, ξ (where $\xi = \frac{(\sigma_1+\sigma_2+\sigma_3)}{\sqrt{3}}$). The Senv yield function is given by

$$f = \overline{\varrho}_{model} - \overline{\varrho}_{actual}, \tag{4.37}$$

where

$$\overline{\varrho}_{actual} = \hat{r}\varrho_{ps_c}$$

and $\hat{r}$ is a function of the Lode angle φ, ϱ_e and ϱ_c such that

$$\hat{r} = \frac{r_1\alpha + \sqrt{2r_1\alpha^2 + r_2}}{2r_1\alpha^2 + 1} \tag{4.38}$$

where

$$\alpha = cos(\varphi + \frac{pi}{6}), \quad r_0 = \frac{\varrho_{ps_e}}{\varrho_{ps_c}}, \quad r_1 = \frac{2(1-r_0^2)}{(2r_0-1)^2} \quad \text{and} \quad r_2 = \frac{r_0(5r_0-4)}{(2r_0-1)}. \tag{4.39}$$

An overbar on a stress measure indicates normalisation with respect to the uniaxial compressive strength, f_c. ϱ_e and ϱ_c are functions of the stress invariant ξ (a measure of a stress state's location along the hydrostatic axis in principal stress space), see Figure 5.29, and calculated using six dimensionless positively valued material constants. Each of these constants can be determined from the following experimental data: uniaxial compression (f_c), uniaxial tension (f_t), equal biaxial compression ($\overline{f}_{bc}$), hydrostatic tension ($\overline{f}_{ht}$) and high level triaxial confinement on the compression meridian ($\overline{\xi}_{tc}$,$\overline{\varrho_c}$). To generate this model stresses (f_c, f_t, f_{bc} and f_{ht}) have a positive value irrespective to whether they are tensile or compressive, but all other stress measures respect a tension-positive sign convention. This function generates a smooth locus with 6-fold symmetry in the deviatoric planes that remains convex when $\hat{r}\left(\frac{\pi}{6}\right) > \frac{1}{2}$.

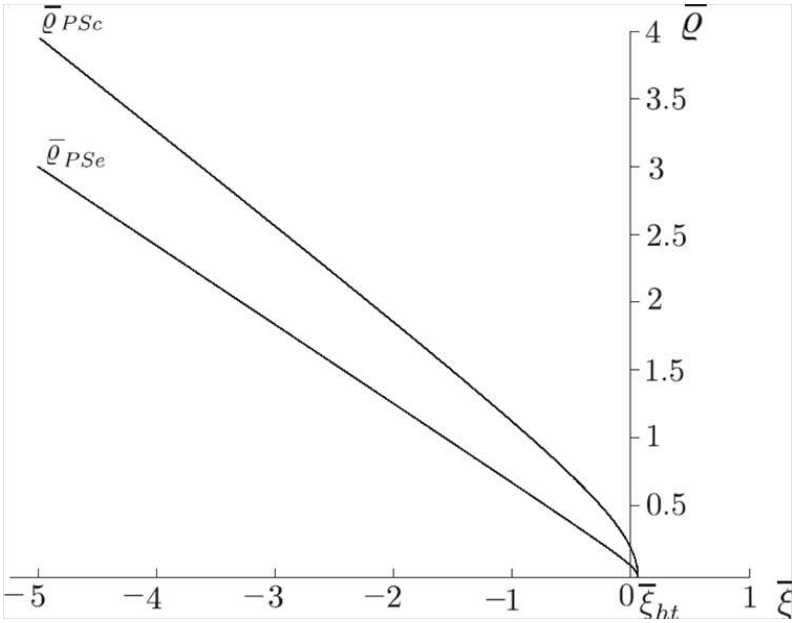


Figure 4.28: Meridional section for Senv

4.5.3 Comparing TCenv, MPenv and Senv

The plot of ϱ against ξ has been used to visualise the relationship between the three envelopes. The extension and compression meridion, ϱ_e and ϱ_c, have been plotted in 5.29 above. Senv would lie between these two meridions, the precise location would depend on the load angle φ. In Figure 4.29, the principle stress axis σ_1 and the axis where $\sigma_2 = \sigma_3$ have been plotted as dotted lines. TCenv has been plotted parallel to σ_1 as a dashed line and MPenv has been plotted as the dot and dashed line. MPenv initial lies on top of TCenv but when the stress state is no longer purely tensile the envelopes separate. This plot can be used to understand the plots in Chapter 5 (Figues 5.28 - 5.30).

TCenv has the largest ϱ values in the regions where the stress states have a tensile and compressive component. Therefore it provides the least conservative safety criterion under this loading condition.

MPenv cuts below ϱ_c on the plot, having the smallest ϱ values in the regions where the stress states have a tensile and compressive component. Therefore it provides the most conservative safety criterion under this loading condition. In the purely tensile region MPenv and TCenv extends further along the hydrostatic axis (the envelope cuts the x axis at a higher value of ξ) and these envelopes have high values of ϱ than Senv. These envelopes therefore provide a less conservative safety criterion under this purely tensile loading condition.

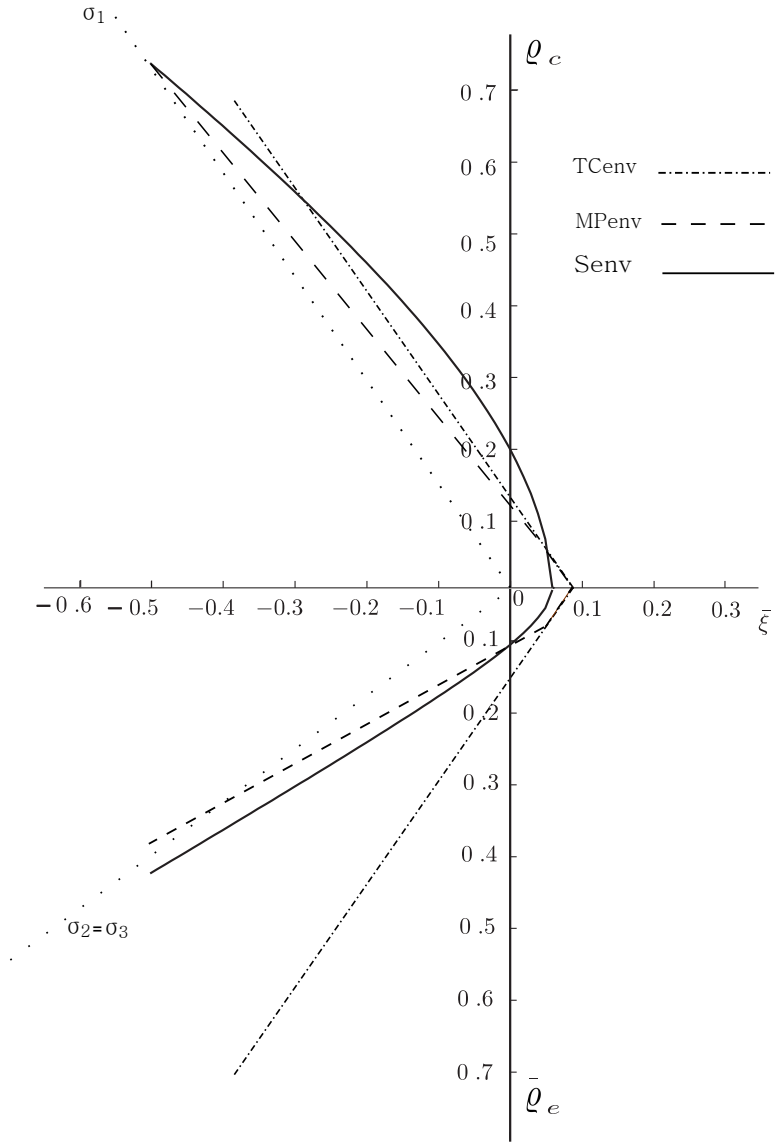


Figure 4.29: Meridional section for Senv, MPend, TCenv

4.5.4 Introduction to `Gmsh`

`Gmsh` [8] is a 3D FE mesh generator with built-in pre- and post-processing facilities. This software is freely available to download online. `.msh` files have be used to view the BCU meshes (original and deformed) using hexahedral elements and generate principal stress plots using line elements. `.msh` files have also been

used to generate the error in energy norm plots, the peak stress envelope plots and coloured principal stress plots where elements are coloured to highlight specific areas of the mesh by specifying element data (see Section 5.6.2).

4.5.5 Principal stress plots

Principal stresses are the three stress vectors normal to the principal planes within the stressed body where there are no shear stresses. These stresses can be defined by principal stress values and principal stress direction vectors. Principal stresses have been calculated from the stress vectors stored at each Gauss point. When FEA is carried out the initial output is nodal displacement. Within `ParaFEM` these displacements are stored in `eld_pp`. Using these displacements, stresses can be recovered at the Gauss points and stored in `tensor_pp`. This is carried out by using the following `ParaFEM` subroutines shape_der, invert and beemat and the `f90` intrinsic function `MATMUL`, in the following order,

```
CALL shape_der(der,points,i)
jac=MATMUL(der,g_coord_pp(:,:,iel))
CALL invert(jac)
deriv=MATMUL(jac,der)
CALL beemat(deriv,bee)
eps=MATMUL(bee,eld_pp(:,1))
sigma=MATMUL(dee,eps)
```

`der` contains the shape function derivatives with respect to the local co-ordinate system, `points` contains the Gauss point locations, `g_coord_pp` contains the global nodal co-ordinates stored on each processor, `jac` is the Jacobian, `deriv` contains the shape function derivatives with respect to the global co-ordinate system, `bee` is the strain-displacement matrix, `eps` is the strain vector and `dee` is the constitutive matrix. `tensor_pp` is a `ParaFEM` array which stores the Gauss point stresses as six component column vectors. These vectors can be outputted to a text file. During post processing, `MATLAB` has been used to calculate the principal stresses and their direction vectors. The stresses were stored as a 3x3 matrix and the principal stresses and their directions were calculated. The `MATLAB` function `[V, D]=eig(sigma)` was used to calculate these values, where `V` is the eigen values (principal stresses), `D` is the eigen vectors (principal directions) and `sigma` is the 3x3 matrix. The principal stress vectors have been plotted as lines proportional to the size of the principal stresses (the three principal stresses intersect at each Gauss point). Principal stress plots have been produced using `.msh` files. Tensile and compressive stress were distinguished using blue for tensile stresses and red for compressive stresses (see Section 5.6.4).

Chapter 5

Nuclear reactor BCU: specific application for the developed ParaFEM code

In this chapter the BCU will be described in detail. The BCU meshes and loading conditions will be introduced. The linear solvers discussed in Section 3.2.2 will be compared when analysing the BCU and instructions will be given on how to refine the meshes further. The results for the BCU FEAs will finally be presented and discussed.

5.1 The boiler closure unit

The Heysham I and Hartlepool reactors comprise upright cylindrical vessels with a multi-cavity design in which the boilers are housed in eight vertical cylindrical cavities, each of which is sealed at its top end by a boiler closure unit and at the bottom by a gas circulator assembly, see Figure 5.1. The boiler closure units are essentially pre-stressed concrete cylinders with nine major vertical cylindrical steam/feed penetrations and two minor inspection and instrumentation penetrations, see Figure 5.2. Each cylinder has an overall diameter of $3.37m$ (including the casing) and depth of $1.73m$. The BCUs are pre-stressed by circumferential wire windings contained within an outer steel casing. The casing is designed to provide a gas tight environment and limit the moisture movement in the concrete, thus protecting the wire windings.

Independent layers of pre-stressing are provided. The active pre-stressing system consists of three layers of $2.6mm$ diameter wire, each initially stressed to 69% of their Guaranteed Ultimate Tensile Strength (GUTS). The active system was designed to maintain a tension-free stress field in the concrete under normal operating conditions, and to provide a minimum ultimate load factor of three times the design pressure. An additional passive pre-stressing system, which comprises six layers of $2.6mm$ diameter wire (each initially stressed to 18% of their GUTS) is also provided. This passive pre-stress was designed to be independently capable of ensuring a minimum ultimate load factor of three. Each layer of wire was

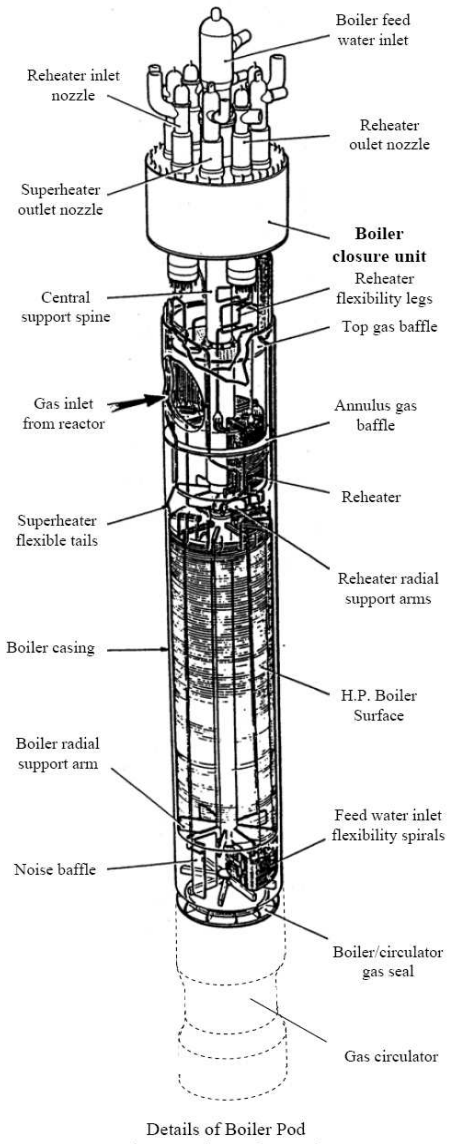


Figure 5.1: Hartlepool/Heysham I PCPV boiler details [1]

wound through approximately six hundred turns onto an outer steel membrane, with the three active layers being wound initially, followed by the six passive layers. Grease was applied to each layer of wire prior to fitting the outer steel casing, see Figure 5.3. The gap between the wires and outer casing was then filled with Fillite thermal insulation and the casing closed by a continuous seal weld around its periphery.

Two independent holding down systems are provided to ensure that the closure unit is secured to the pre-stressed concrete pressure vessel. Each system is designed with a factor of safety of three times the design pressure. The primary system consists of forty-eight, $69mm$ thread diameter, holding down bolts which pass through the full depth of the closure within steel tubes; anchoring the closure to the boiler

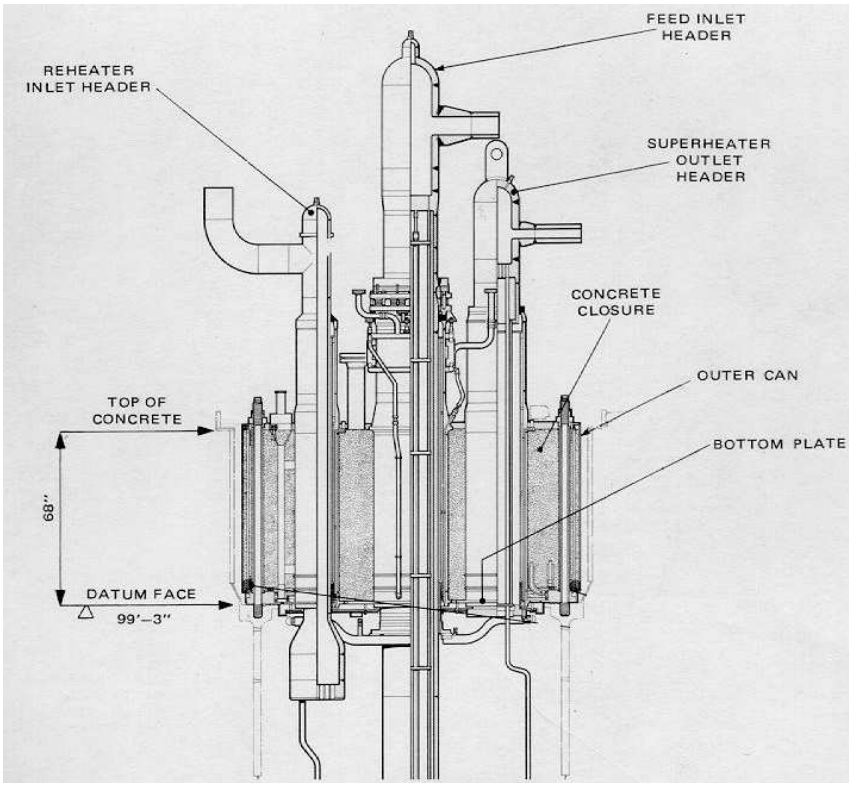


Figure 5.2: Hartlepool/Heysham I PCPV BCU: Sectional elevation. The concrete is identified by the darker grey shading. [1]

Figure 5.3: Photograph of pre-stressed wires being wound around the BCU [1]

liner closure flange. The gas tight connection between the closure unit and boiler liner flange is achieved using O-ring seals. The secondary system, which forms a restraint in the unlikely event of failure of the primary system, comprises steel shoes which transfer the gas load to a reinforced concrete back-up ring,

located above the closure, which is anchored to the top of the PCPV by thirteen vertical tendons.

Each major penetration comprises of three concentric cylinders which form a shutter tube, inner sleeve (water jacket tube) and insulation sleeve. On the gas side face of the closure unit, a bottom liner plate is welded to the bottom seal ring and the penetration shutter tubes to form a gas tight membrane. The total weight of each boiler is supported from the closure unit by a vertical spine which is attached to the boiler closure via the centrally located feed penetration.

The penetration sleeves and bottom liner plate are insulated against boiler gas temperatures, and a cooling system is provided in the form of water jackets to the penetrations and cooling water pipes welded to the bottom liner plate and seal ring. The shutter tubes of the penetrations and the liner plate are constructed from mild steel.

5.2 BCU mesh generation

An 8-noded hexahedral element mesh of one quarter of the BCU was made available at the start of the project. It was originally created by the UK consulting engineers undertaking a safety analysis. That mesh has been refined using the Fortran 90 program `mesh_refine.f90`. The mesh refinement strategy converts the 8-noded elements initially into 20-noded elements before subdividing each element into eight 8-noded elements. The procedure can then be repeated. The following meshes have been generated for the BCU: 8(8-noded elements), 20(20-noded elements), 88(each 20-noded element is subdivided into eight 8-noded elements) and 820(each 8-noded element within 88 is converted to a 20-noded element). Table 5.1 provides the number of elements and nodes within each mesh.

Mesh	No. of elements	No. of nodes
8	40,201	48,330
20	40,201	185,403
88	321,608	354,536
820	321,608	1,385,616

Table 5.1: No. of elements and nodes in each BCU mesh

5.3 BCU loading conditions

The BCU will experience different loading conditions dependent on whether it is operational or not, and whether the BCU wire windings are intact or not. To model these conditions five possible loading conditions have been considered:

1. underside pressure and associated loading within each penetration
2. bolt loads
3. radial pre-stress due to radial wire windings
4. weight of the boiler
5. self weight

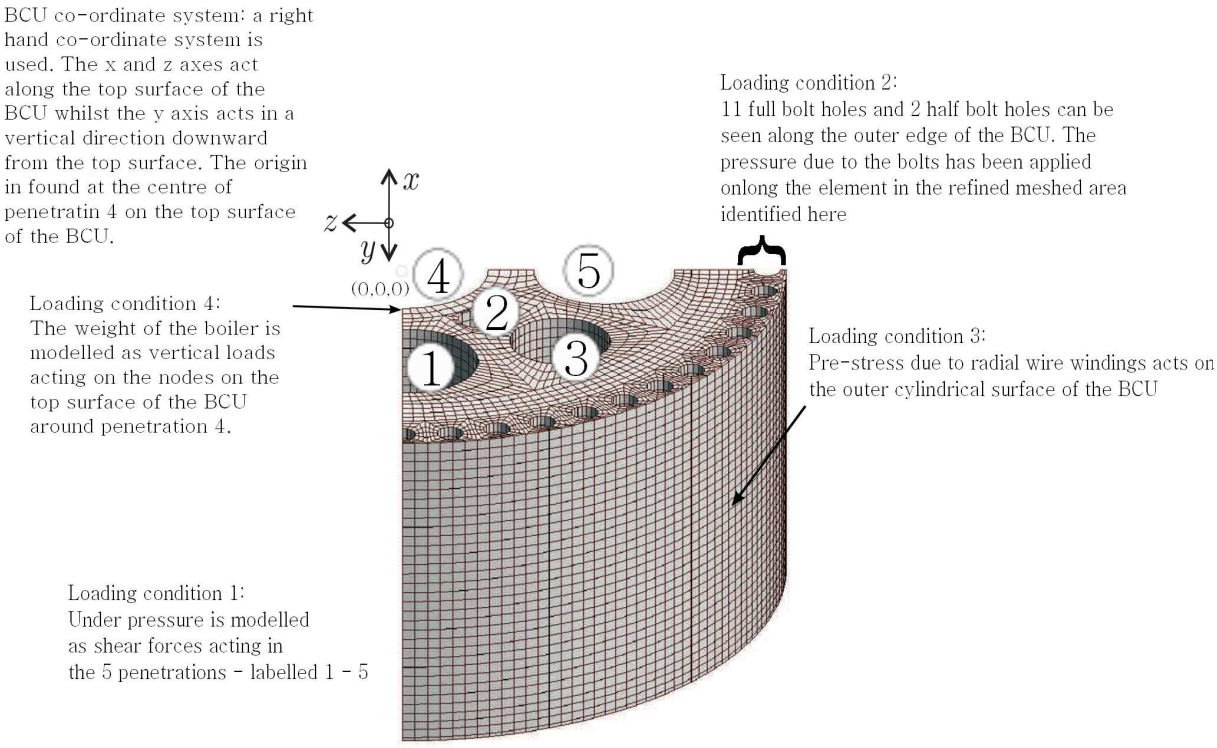


Figure 5.4: BCU mesh showing loading conditions and co-ordinate system used

Figure 5.4 highlights where these loading conditions are action on the mesh and identifies the co-ordinate system used for the BCU. Three load cases have been chosen for the analyses and are modelled, using a combination of the above loads as follows:

Load case 1: pre-operational condition - BCU is pre-stressed but there is no under pressure because the nuclear reactor is not operating

- bolt loads
- radial pre-stress due to radial wire windings
- weight of the boiler
- self weight

Load case 2: normal operational condition - the maximum pressure is applied under normal operating conditions and pre-stressing remains intact

- underside pressure and loading within each penetration
- bolt loads
- radial pre-stress due to radial wire windings
- weight of the boiler

- self weight

Load case 3: operational conditions with loss of pre-stressing - maximum pressure is applied but pre-stressing has been lost

- underside pressure and associated loading within each penetration
- bolt loads
- weight of the boiler
- self weight

A Fortran program `calc_bcu_lds.f90` has been written to generate the loading files `p121.lds` and `p121.prs`. In the following five subsections, the loading conditions are described in more detail. The contribution of each loading condition to the total force will be estimated and checked by running an example BCU problem using `ParaFEM`. Similar checks were carried out for each load case for each mesh. Areas, the circumference and the depth have all been estimated using the BCU nodal co-ordinates.

5.3.1 Underside pressure and associated loading within each penetration

The pressure (p_u) applied to the underside of the BCU is $3.6MPa$. This pressure acts on all element faces in the $x-z$ plane at $y=0$. This surface has a total area A_u. To verify that this pressure is being applied correctly to this face, the following check was carried out.

A_u = Total area in $x-z$ plane at $y=0$ - cross sectional area of inlets in $x-z$ plane $= A - A_i$
$A = \left(\frac{1}{4} \times \pi \times (1334)^2\right) = 1,397,660mm^2$
$A_i = 21,343 + 141,637 + 145,437 + 145,437 + 102,670 = 556,524mm^2$
therefore,
$A_u = 1,397,660 - 556,524 = 841,136\ mm^2$.

The force in the y direction applied to the BCU due to the pressure acting on its underside (not including the loading within the inlets) can now be determined as

$$\begin{aligned} F_{u_y} &= p_u A_u \\ &= 3.6 \times 841,136 = 3.028MN. \end{aligned}$$

When this pressure was the only loading applied to the BCU, for the 8-noded mesh the total applied force in the vertical direction was, $F_y = -2.983MN$.

The underside pressure also acts within the penetrations and can be modeled as upward forces acting around the circumference of the penetrations. The nodal forces must be equivalent to $p_u = 3.6MPa$ acting over the area of the inlets A_i in the $x-z$ plane. The nodal forces have been calculated by determining the upward force associated with each penetration and then applying a fraction of the force to each node. The nodal forces around a penetration are directly proportional to the length of the

element edges associated with each node around the circumference (or arc length) of the penetration (dependant on whether penetration is at a BCU boundary or not).

The force in the y direction applied to the BCU due to the pressure acting on its underside acting within the penetration only, can now be estimated as

$$\begin{aligned} F_{i_y} &= p_i A_i \\ &= 3.6 \times 556,524 = 2,003,486N. \end{aligned}$$

When these nodal forces were the only loading condition applied to the BCU, for the 8-noded mesh the total applied force in the vertical direction was, $F_y = -2.003MN$.

Having undertaken some preliminary analyses, it became apparent that applying nodal forces around the top circumference of the penetrations causes more local deformation of the concrete than is realistic, see Figures 5.5. The ridges around each penetration distort the error analysis due to the

Figure 5.5: Deformed mesh with nodal ring loading due to maximum underside pressure (deformation scale factor 200)

unrealistically high local loading in these areas. The loading conditions were reconsidered and a linearly varying shear force was applied across the element surfaces within each penetration. The shear force varied from zero at the underside to a maximum shear force $\sigma_{xy_{max}}$ at the top. The total force exerted due to these shear forces was F_{i_y} as calculated above.

The vertical shear stress acting within the penetration follows a linear variation,

$$\sigma_{xy} = \sigma_{xy_{max}} \frac{z}{d}. \tag{5.1}$$

To calculate $\sigma_{xy_{max}}$, the force acting on the elements within the penetration is given by the integral of the shear stress acting over the area inside the penetration, where the circumference of the penetration, c, the length on the penetration, d, and the total force due to the underside pressure, F_i are known.

$$F_{i_y} = \int_0^d c\sigma_{xy_{max}} \frac{z}{d} dz. \tag{5.2}$$

The only unknown in (5.2) is $\sigma_{xy_{max}}$. If the integration is carried out, an expression for $\sigma_{xy_{max}}$ is obtained,

$$F_i = \left[\frac{c}{d}\sigma_{xy_{max}} \frac{z^2}{2}\right]_0^d \tag{5.3}$$

$$= \frac{c}{2d}\sigma_{xy_{max}}(d^2 - 0) \tag{5.4}$$

$$\sigma_{xy_{max}} = \frac{2F_i}{cd} \tag{5.5}$$

Once $\sigma_{xy_{max}}$ is obtained σ_{xy} can be applied to the nodes inside the penetration using (5.1). The shear stresses are converted to nodal forces using the `prs2nforce` subroutine, see Section 4.2. The deformed meshes shown in Figure 5.6 show the improved realism around the penetrations.

Figure 5.6: Deformed mesh with linearly varying shear forces acting inside the penetrations (deformation scale factor 200)

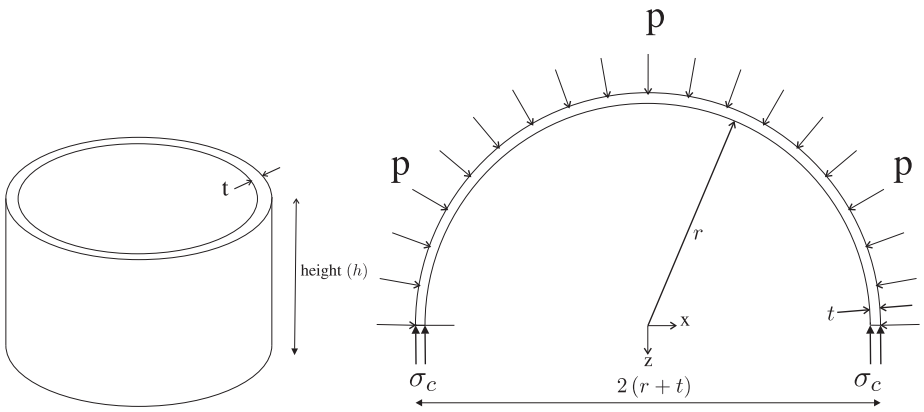


Figure 5.7: Forces acting on the BCU due to the pre-stressed wire windings

5.3.2 Bolt loads

Each of the twelve bolts running the length of the BCU exerts a vertical compressive force of $974kN$. The total force due to the bolts is $11.688MN$. This is modelled as a pressure applied to the upper face of the BCU around the bolts; that is on the elements within the refined meshed area around each bolt. This area, $A_b = 369,035mm^2$ implies that the pressure applied due to the bolt loads is $p_b = \frac{11,688,000}{369,035} = 31.7MPa$.

When this pressure was the only loading applied to the BCU, for the 8-noded mesh, the total applied force in the vertical direction was, $F_y = 11.780MN$.

5.3.3 Pre-stress due to radial wire windings

Pre-stressed wires were wrapped around the circumference of the BCU. There are nine layers of closely-packed wires. The total pressure exerted by these wires on the BCU can be calculated by summing the pressure contribution from each wire layer $i = 1, 9$ over a height h, where the wire has a thickness, t_i. By considering equilibrium, the equivalent pressure exerted on the BCU can be calculated for each wire winding,

$$\begin{aligned} F_y = p\,(2\,(r+t))\,h &= 2th\sigma_c \\ p\,(r+t) &= t\sigma_c \\ p &= \frac{t_i\sigma_c}{r_i+t_i}. \end{aligned} \tag{5.6}$$

When nine wires are considered,

$$P = \sum_{i=1}^{9} \frac{t_i\sigma_c}{r_i+t_i}. \tag{5.7}$$

The pre-stress wire contributions have been calculated and listed in Table 5.2,
The total pressure exerted onto the outer cylindrical surface by the radial wire windings is $p_w = 8.65MPa$. This pressure can be considered to have two components: one acting in the x direction and another in

Layer i	$t_i\,(mm)$	$r_i\,(mm)$	$\sigma_c\,(MPa)$	$P_i\,(MPa)$
1	2.6	1616.1	1185	1.9034
2	2.6	1618.7	1185	1.9004
3	2.6	1621.3	1185	1.8973
4	2.6	1623.9	310	0.4955
5	2.6	1626.5	310	0.4948
6	2.6	1629.1	310	0.4940
7	2.6	1631.7	310	0.4932
8	2.6	1634.3	310	0.4924
9	2.6	1636.9	310	0.4916

Table 5.2: BCU pre-stress wire loading contributions

the z direction. The area on which these pressure components act is $A_{w_{x,z}} = 2,586,083mm^2$, therefore the force components acting in the x and z directions can be determined as

$$\begin{aligned} -F_{w_x} &= p_w \times A_{w_x} \\ &= 8.65 \times 2,586,083 = 22.370MN \\ F_{w_z} &= p_w \times A_{w_z} \\ &= 8.65 \times 2,586,083 = 22.370MN. \end{aligned}$$

When this pressure was the only loading condition applied to the BCU, for the 8-noded mesh the total applied force in the x and z directions were,

$$\begin{aligned} -F_{w_x} &= 22.370MN \\ F_{w_z} &= 22.370MN \end{aligned}$$

5.3.4 Weight of boiler

The weight of the boiler is $100kN$. This is modeled as a vertical load acting downwards from the central penetration. The total load is shared between the nodes proportional to the length of element associated with the node. Only $\frac{1}{4}$ of the BCU is being considered, therefore $25kN$ is distributed between the nodes. The total vertical force due to the weight of the boiler is $F_{y_b} = 25kN$. When these nodal forces were the only loading conditions applied to the BCU, for the 8-noded mesh the total applied force in the y direction was

$$F_{y_b} = 25kN.$$

This load is very small compared to the other loads acting on the BCU.

5.3.5 Self weight

The BCU is a concrete structure with a density, $\rho = 2.4 \times 10^{-6} kgmm^{-3}$. The self weight of the BCU can be estimated by assuming an acceleration due to gravity of $g = 9810mms^{-2}$ and estimating the volume

of the BCU. The estimated volume of the BCU is

$$\begin{aligned} V &= (A_u \times 1600) + (A_b \times 1708) + (A_j \times (1708 - 108)) \\ &= (841136 \times 1708) + (369035 \times 1600) + \left(\frac{\pi}{4}\left(1409^2 - 13334^2\right) \times (1708 - 108)\right) \\ &= 590,456,000 + 1,436,660,288 + 259,946,900 = 2,287,063,188 mm^3. \end{aligned}$$

The self weight of the BCU can therefore be estimated as

$$\begin{aligned} F_{sw_y} &= \rho \times g \times V \\ &= -2.4 \times 10^{-6} \times 9810 \times 2,287,063,188 \\ &= -5.385 MN. \end{aligned}$$

When the self weight of the BCU was the only loading condition considered, for the 8-noded mesh the total applied force in the vertical direction was

$$F_{sw_y} = -5.345 MN.$$

A summary of the loading conditions for each mesh is provided in Table 5.3.

Mesh	Load case 1		Load case 2		Load case 3	
	Nodal loads	Pressurised faces	Nodal loads	Pressurised faces	Nodal loads	Pressurised faces
8	15	2160	15	7715	15	6131
20	29	2160	29	7715	29	6131
88	29	8640	29	30860	29	24524
820	57	8640	57	30860	57	24524

Table 5.3: Summary of the number of nodal loads and pressurised faces for each BCU load case and mesh

5.4 Using different linear solvers for BCU FEAs

As described in Section 3.2.2 two parallel linear solvers can be run within `p121.f90`: the PCCG solver and `MUMPS`. Here we will present a run time comparison for the two solvers, taking the mean of three run times for all meshes, when carrying out the analysis for load case 2. A tolerance of 0.1E−7 was used for the PCCG solver (note that this tolerance the solutions outputted by the PCCG solver matched those provided by the direct solver, to the nearest $0.0001mm$). Run times have been recorded using the function `elap_time()`, however it can be observed that run times are not very repeatable. This could be due to allocation of different *Hamilton* nodes or interference from other *Hamilton* jobs slowing down communication between processors. To solve this problem specific nodes (see reference to computer nodes in Section 3.1) would have to be reserved for the jobs. However general trends in run times can be identified and conclusions drawn.

5.4.1 BCU FEAs run time comparisons

It was hoped that the multi-frontal strategy adopted by MUMPS would have the potential to solve larger problems more quickly than the PCCG solver on parallel machines. Speed gain was seen using MUMPS, however parallel speedup when using more processors was very limited, see Table 5.4. When running analyses for load case 2 on Mesh 8, 20 and 88 the run times were quicker when using MUMPS and up to 4 nodes, however when the number of nodes increased from 8 to 16 the run times decreased most significantly when using the PCCG solver. For the coarser meshes (Mesh 8 and 20) the analysis using 16 nodes was still quickest when using MUMPS - 22 seconds and 239 seconds for the two meshes compared with 40 seconds and 377 seconds when using the PCCG solver. However for a more refined mesh, Mesh 88, there was not only greater speed up when using the PCCG solver but when 16 nodes where used the PCCG solver also carried out the analysis twice as quickly as MUMPS (2436 seconds using MUMPS and 1182 seconds using the PCCG solver. MUMPS also carried out the analysis 4 times slower when using 16 nodes compared to 1 node. When carrying out analyses on the larger meshes, Mesh 88 and 820, MUMPS often struggled to complete the analysis. This was due to memory shortages. This meant that results were not obtained using MUMPS for Mesh 820 and results were only obtained for Mesh 88 when using 1 to 8 nodes. The PCCG solver however continued to show speed up when using more nodes for these larger meshes. For all the meshes there was an increase in run time when using 16 nodes in comparison to 8 nodes (except for Mesh 8 when using MUMPS). There were also some results where the run time was significantly increased when using 8 nodes. Both of these findings could be explained by the fact that cores on Hamilton either consist of 4 or 8 nodes. When analyses require more than one core the analysis could be slowed down due to communication between cores. Unfortunately investigation did not fully allow the underlying problems to be identified or solved, and although MUMPS did provide a faster analysis for Meshes 8, 20 and 88 when using 1 to 4 nodes, this solver's performance was worse than the the PCCG solver for the finer meshes especially when using more than 8 nodes.

5.5 Running a BCU parallel FEA

5.5.1 Steps required to further refine the BCU mesh

In the following, the instructions are given on how to further refine the BCU mesh and generate the required input files. Three Fortran programs have been written to allow this to be accomplished, `mesh_refine.f90`, `find_bcu_bnd` and `cal_bcu_ld.f90`. The steps are as follows.

1. Use `mesh_refine.f90` to obtain new element topology (`etpl`) and global co-ordinates (`coord`) and generate the input file `p121.d`. To run `mesh_refine.f90` correctly the input and output file names must be altered and the correct subroutines must be commented in or out of the program. For example, to convert an 8-noded mesh to a 20-noded mesh, the subroutine `convert8to20nod` must be included in the main program whilst if the conversion is from a 20-noded mesh to an eight 8-noded mesh the subroutine `convert20to88nod` must be included in the main program. The correct input must be entered into the subroutines `read_data` and `write_input_files`.

2. Use `find_bcu_bnd.f90` to locate the boundary conditions for the BCU and generate `p121.bnd`. To run `find_bcu_bnd.f90`, the variables `nn` (number of nodes) and `ne` (number of elements) must be

Mesh	Solver	Number of cores	Solver run time /s
8	MUMPS	1	120
		2	45
		4	31
		8	153
		16	22
	PCCG	1	177
		2	92
		4	48
		8	32
		16	40
20	MUMPS	1	274
		2	235
		4	241
		8	1203
		16	239
	PCCG	1	1799
		2	968
		4	666
		8	324
		16	377
88	MUMPS	1	589
		2	547
		4	681
		8	2436
	PCCG	1	4365
		2	3789
		4	2111
		8	1110
		16	1182
820	PCCG	8	45883
		16	15692

Table 5.4: Linear solver run time comparisons for BCU FEA (load case 2)

correct and the corresponding `p121.d` and `p121.dat` files must be stored in the same folder.

3. Use `calc_bcu_ld.f90` to calculate the BCU's loading conditions and generate `p121.prs` for the pressure loading and `p121.lds` for the point loads. To run `calc_bcu_ld.f90`, the variables `nn`, `ne` and `nod` (number of nodes per element) must be correct and the corresponding `p121.d` and `p121.dat` files must be stored in the same folder. The variables `pre-stress` and `maxP` must take the value 1 or 0. When `pre_stress`= 1, the pre-stressing is undamaged but if `pre-stress`= 0 the pre-stressing is no longer present. When `maxP`= 1, the under-pressure on the BCU is at operational level, but if `maxP`= 0 then the BCU is is not subject to any under-pressure. Combinations of these variables allow the files for the three loading cases to be generated.

5.6 Results

Results have been processed from the FEAs carried out on the four meshes for the three different load cases. In this section the results are presented and discussed by considering the displacements, principal stresses, peak stress envelopes and error analyses.

5.6.1 Displacements

The displacements of the same three nodes have been recorded for each mesh as a displacement vector $\{u \quad v \quad w\}^T$ and also in terms of radial d_r and tangential movement d_t, where a positive radial movement indicates the BCU node in question is being squeezed inwards towards the BCU origin and a positive tangential movement indicates the BCU node is moving to the right if the BCU is viewed down the radius from the node to the origin, see Figure 5.8. The location of the three nodes have been highlighted in Figure 5.8. The displacements are recorded in mm at each node (for the four meshes and three load cases) in Tables 5.5, 5.6 and 5.7. Figures 5.9, 5.10 and 5.11 show how the meshes deform under the different load cases. The displacement of each node is scaled by a factor of 400, then added to the original global co-ordinates to obtain the new exaggerated plot co-ordinates. These highly exaggerated deformation plot indicate how the BCU distorts under the different loading conditions.

The weight of the boiler causes localised large deformations in the z direction across the top row of elements around the inner penetration under load case 1. The bolt loads cause significant deformation across the top of the BCU and very considerable distortion around the step on the underside of the BCU. The actual structure however had a steel plate across the step which will have reduced the distortion. The effect of the underside pressure and associated penetration shear stresses can be seen in load case 2. The BCU is pushed upwards. This reduces the effect of the distortion due to the weight of the boiler. The distortion due to the bolt loading is also altered. The pinching-in on the outer edge of the BCU (seen in the deformation plots for load case 1) is reduced and less distortion is seen around the underside step. Most striking is the change that occurs between load cases 2 and 3. When the pre-stressing is removed from around the circumference of the BCU, the BCU expands outwards and the the structure is seen to bulge more across its top surface.

These observations are backed up by the numbers in the displacement tables (Tables 5.5, 5.6 and 5.7). The radial movement for each mesh and for each of the three nodes shows how, under load cases 1 and 2, the pre-stressing causes the node to move inwards towards the origin of the BCU (a positive displacement), however when the pre-stressing is released, under load case 3, the node moves outwards. The magnitude of the radial displacements can also be observed. The maximum displacement for node 795 is seen under load case 1 ($\approx 0.3mm$). The maximum displacement for node 560 is seen under load case 3 when the pre-stressing is removed ($\approx 0.5mm$). The maximum displacement for node 6956 is seen under load cases 1 and 2 when the pre-stressing is in place ($\approx 0.6mm$).

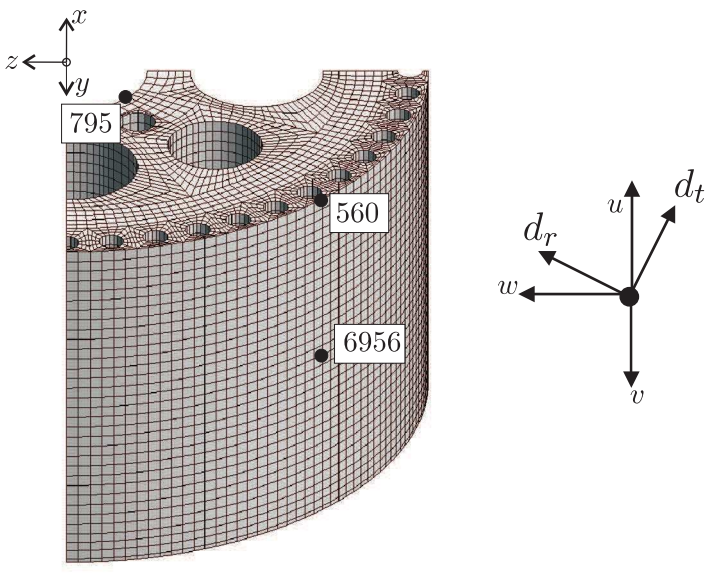


Figure 5.8: Identification of three BCU nodes and radial and tangential displacement directions

Node 795

Original coordinates $(255.7, 1708.0, -255.7)$

Mesh	Load case	$u\ mm$	$v\ mm$	$w\ mm$	$d_r\ mm$	$d_t\ mm$
8	1	-0.1757	-0.1819	0.1991	0.2650	0.0165
	2	-0.0507	0.4201	0.0621	0.0798	0.0081
	3	0.1422	0.2966	-0.1600	-0.2137	-0.0126
20	1	-0.1941	-0.2159	0.2140	0.2886	-0.0142
	2	-0.0509	0.4226	0.0617	0.0796	0.0076
	3	0.1448	0.3014	-0.1622	-0.2171	-0.0123
88	1	-0.1875	-0.2069	0.2088	0.2801	0.0151
	2	-0.0509	0.4219	0.0619	0.0797	0.0078
	3	0.1440	0.3001	-0.1616	-0.2160	-0.0125
820	1	-0.2044	-0.4389	0.2229	0.3021	0.0059
	2	-0.0508	0.4227	0.0616	0.0794	0.0076
	3	0.1450	0.3017	-0.1623	-0.2173	-0.0122

Table 5.5: Displacement recorded in mm for node 795 of the BCU meshes

Node 560

Original coordinates $(1143.0, 1708.0, -1143.0)$

Mesh	Load case	$u\ mm$	$v\ mm$	$w\ mm$	$d_r\ mm$	$d_t\ mm$
8	1	-0.2393	-0.6509	0.2438	0.3417	0.0032
	2	-0.1316	-0.5586	0.1346	0.1882	0.0021
	3	0.3432	-0.7534	-0.3449	-0.4865	-0.0012
20	1	-0.2473	-0.6526	0.2518	0.3529	0.0032
	2	-0.1393	-0.5586	0.1426	0.1993	0.0023
	3	0.3434	-0.7551	-0.3452	-0.4869	-0.0013
88	1	-0.2453	-0.6534	0.2498	0.3501	0.0032
	2	-0.1371	-0.5589	0.1403	0.1962	0.0023
	3	0.3434	-0.7546	-0.3452	-0.4869	-0.0013
820	1	-0.2523	-0.6404	0.2606	0.3627	0.0059
	2	-0.1404	-0.5582	0.1436	0.2000	0.0023
	3	0.3431	-0.7551	-0.3448	-0.4864	-0.0012

Table 5.6: Displacement recorded in mm for node 560 of the BCU meshes

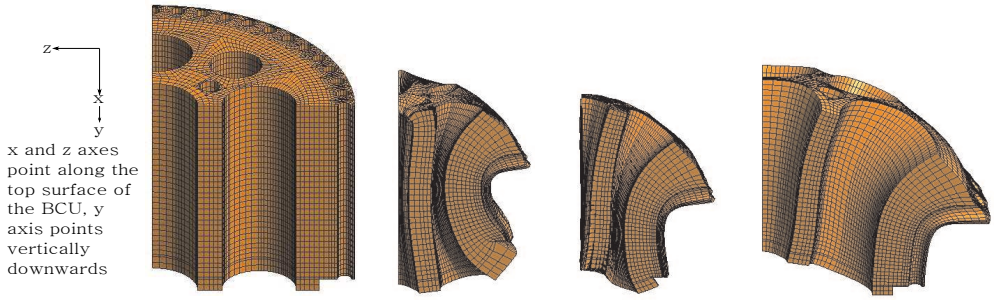
Node 6956

Original coordinates $(1143.0, 855.7, -1143.0)$

Mesh	Load case	$u\ mm$	$v\ mm$	$w\ mm$	$d_r\ mm$	$d_t\ mm$
8	1	-0.4444	-0.1980	0.4487	0.6316	0.0031
	2	-0.4451	-0.1189	0.4500	0.6329	0.0034
	3	0.0007	-0.2109	-0.0007	-0.0009	0.0000
20	1	-0.4508	0.2004	0.5552	0.6406	0.0031
	2	-0.4513	-0.1198	0.4562	0.6417	0.0035
	3	0.0017	-0.2116	-0.0012	-0.0020	0.0003
88	1	-0.4491	-0.2007	0.4535	0.6414	0.0030
	2	-0.4496	-0.1196	0.4545	0.6425	0.0036
	3	0.0014	-0.2114	-0.0012	-0.0018	0.0001
820	1	-0.4514	-0.2001	0.4556	0.6414	0.0030
	2	-0.4518	-0.1200	0.4568	0.6425	0.0036
	3	0.0017	-0.2117	-0.0014	-0.0022	0.0002

Table 5.7: Displacement recorded in mm for node 6956 of the BCU meshes

The results recorded in Tables 5.5, 5.6 and 5.7 indicate that the coarse mesh provided a (perhaps surprisingly) good estimate of the displacements at 3 particular nodes singled-out for examination. The difference between the run-times from mesh 8 and mesh 820 is dramatic (the later takes 700 times longer to complete, when using the PCCG solver with 16 cores for load case 2 - see Table 5.4 for run time comparisons) whereas the displacements in the radial direction (d_r) differ by less at $0.01mm$ at node 6956 (out of a maximum of $\approx 0.64, 0.64$ and $0.002mm$ for load cases 1, 2 and 3 respectively). The difference in displacement at the other two nodes is just as surprising: at node 795 the difference is less than $0.04mm$ (out of a maximum of $\approx 0.30, 0.08$ and $0.22mm$) whilst at node 560 the difference is less than n$0.03mm$ (out of a maximum of $\approx 0.36, 0.2$ and $0.49mm$).

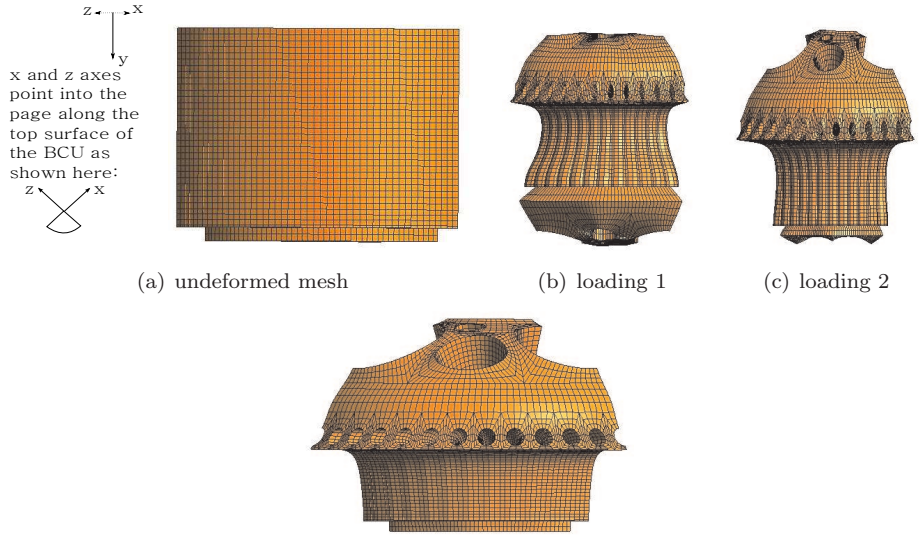


(a) undeformed mesh (b) loading 1 (c) loading 2 (d) loading 3

Figure 5.9: Exaggerated deformation plots: view 1 (deformation scale factor 400)

(a) undeformed mesh (b) loading 1 (c) loading 2 (d) loading 3

Figure 5.10: Exaggerated deformation plots: view 2 (deformation scale factor 400)



(a) undeformed mesh (b) loading 1 (c) loading 2

(d) loading 3

Figure 5.11: Exaggerated deformation plots: view 3 (deformation scale factor 400)

Load case	1			2			3		
Mesh	Senv	MPenv	TCenv	Senv	MPenv	TCenv	Senv	MPenv	TCenv
8	921	2572	1236	851	2517	944	8351	10922	9552
20	1838	2764	1621	1949	2521	1155	8657	11477	9951
88	6733	15610	7255	7346	15684	5468	64259	82878	72227
820	10005	16324	8174	11337	16273	6260	65071	85004	73775

Table 5.8: Table to show the number of elements with principal stresses lying outside the peak stress envelopes

5.6.2 Peak stress envelopes

Here the FEA results are examined to determine how the release of the pre-stressing (due to the circumferential wires corroding) would affect the structural integrity of the BCU. Three different peak stress envelopes are used: Senv, MPenv and TCenv, as introduced in Section 4.5.1. The effect of refining the BCU mesh will be discussed and a comment made on how non-linear analysis could affect the results if it was incorporated in the code.

Peak stress envelope results

The number of elements within each mesh containing at least one Gauss point stress lying outside the peak stress envelopes have been recorded in Table 5.8 for each of the load cases. The number of elements where this is the case increases significantly when the pre-stressing is released (load case 3). The number of elements lying outside each peak stress envelope increases from approximately 2% to 24% for Senv, approximately 6% to 27% for MPenv, approximately 2% to 21% for TCenv when considering Mesh 8. Figures 5.12 - 5.23 show these elements highlighted in green for all four meshes. Figures 5.24 - 5.27 present similar results however information for all three peak stress surfaces is provided on the same mesh allowing comparisons to be made. The figures can be understood by looking at the key provided in Figure 5.28.

Figures 5.12, 5.15, 5.18 and 5.21 show that under load case 1 elements located at the top of the inner penetration, some elements within each penetration and around the bolts contain stress states which lie outside the peak stress envelopes. When load case 2 is considered (see Figures 5.13, 5.16, 5.19 and 5.22) the elements around the top of the inner penetration no longer contain stress states which lie outside the peak stress envelopes, but the number of these elements on the surface of the other penetrations increases. Finally when load case 3 is considered (see Figures 5.14, 5.17, 5.20 and 5.23) the number of elements where stress states exceed the peak stress increases significantly, particularly across the top of the BCU and within each of the penetrations.

For all three load cases there are areas of the BCU where stress states are only outside MPenv or Senv (but inside Senv and TCenv or MPenv and TCenv respectively). There are never any areas where stresses are only outside TCenv (but inside Senv and MPenv). TCenv is the least conservative of the three envelopes, this is because it does not consider the multi-axial behaviour of the concrete (as explained in Section 4.5.1).

For load cases 1 and 2, TCenv and MPenv indicate over-stressed areas around the bolts and the underside-step of the BCU. Senv does not pick up as many elements around the bolts where the gauss point stresses exceed the peak stress, however some over-stress is captured. All three envelopes indicate over-stressed elements around the inner penetration for load case 1. This is due to the application of nodal loads along the penetration's top elements used to model the weight of the boiler. There are areas within the penetrations which appear to be over-stressed only according to the MPenv and Senv. TCenv does not pick up these areas of over-stress within the penetrations. This can be seen in Figures 5.12 - 5.23 where the green area for load cases 1 and 2 is considerably smaller within the outer penetration on view.

For load cases 3 all three peak stress envelopes predict that a large volume of the BCU will have stress states lying outside the peak stress envelope ($\approx 21\%, 27\%$ and 24% for Senv, MPenv and TCenv respectively when using Mesh 8). MPenv gives rise to extra elements violating the criterion around the edge of the zone of over-stressed elements.

Figure 5.29 allows comparison between Senv and MPenv further. All the stress states which lie outside MPenv have been plotted in ξ, ρ space. This plot shows the compression meridian (ρ_c) and extension meridian (ρ_e) for Senv. Senv will lie between these two meridians. The exact location will depend on the Lode angle. From Figure 5.29 many of the stress states lie between ρ_c and ρ_e. Under load case 2 nearly all the stress states have some compressive component (defined by negative ξ values), however under load case 3 the stress states become dense around $\xi = 0$ and there is a significant proportion which lie in the area of the plot where $\xi > 0$ indicating more of the stress states have at least one tensile component. Since the stress states lie between ρ_e and ρ_c it is not possible to determine whether they lie inside or outside Senv. For more information on whether the stress states lie inside Senv when outside MPenv see Figure 5.30. These figures show how some of the stress states lying outside MPenv fall inside Senv. This could occur when the stress state is a combination of tensile and compressive components. This is an area of interest because concrete can fail even when all three components of the stress state are lower than the tensile and compressive strength of concrete. It is therefore important to model the strength accurately. This is most notable under load case 3, where there are stress states outside Senv but inside MPenv. These are densely plotted close to where $\xi = 0$. Figure 5.31 shows that there are also stress states that lie inside MPenv but outside Senv. The figures (both 5.30 and 5.31) do not tell us the Lode angle of the stress states so it is not possible to determine exactly where these stress states occur.

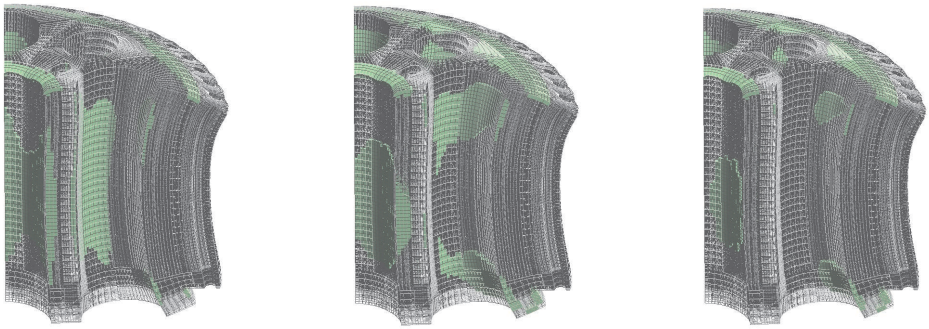
Figure 5.29 shows that some of the stress states have some tensile component under load case 3. The BCU was design to avoid the situation where any of the stresses were tensile. The number of tensile stress states was recorded for Mesh 8 under load case 3. The number of Gauss points where the stress state is tensile and outside MPenv is $21,409$ whereas for Senv the number is $20,828$ (when there are a total of $321,608$ Gauss points in the mesh). This corresponds to approximately 6.7% and 6.5% of the total Gauss point for MPenv and Senv respectively.

The FEA results for the BCU suggests that under extreme conditions when all the pre-stress is lost (load case 3) the BCU's structural integrity is severely compromised. In the analyses (when using the most conservative model) 27% of the Gauss point stresses fell outside the peak strength envelope and

6.7% of the stresses had at least one tensile principal stress component. These results suggest that the BCU would be no-longer safe to operate under these condition because there would be the possibility of concrete rupture within the BCU. However, in reality such an extreme situation where all pre-stress is lost is unlikely to occur. A more realistic situation would be the gradual reduction in pre-stress due to deterioration of individual wires. The results do suggest that when modeling the BCU, care must be taken when using any peak stress envelope in isolation. When the BCU comes under combined tensile and compressive loads the two most accurate envelopes, Senv and MPenv, suggest slightly different failure criterion. MPenv predicts than more elements will be over-stressed. This thesis does not address which of these envelopes provides the most accurate estimation but rather highlights the need to be careful when considering the strength of concrete under these conditions. The results also show that care must be taken when using TCenv for the analysis of concrete structures under combined tensile and compressive loading.

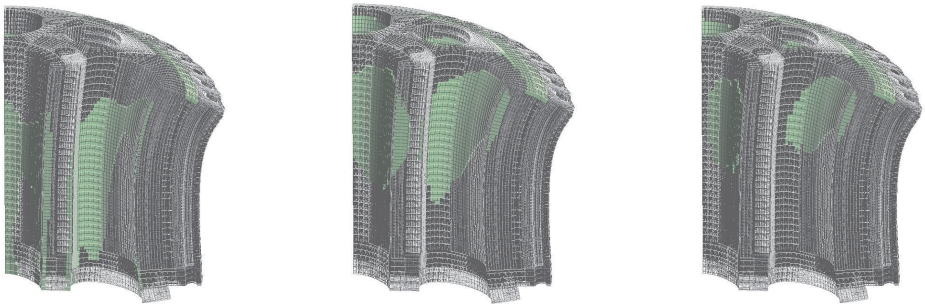
5.6.3 How non-linear FEA could alter the findings on the structural condition of the BCU

Non linear FEA would almostly certainly result in more Gauss point stresses reaching the peak stress critera. When non linear FEA is carried out, the stress states are prohibited from lying outside the yield surface. This means that where stresses meet the yield surface, the self-equilibrating internal body forces in the area of high stress are redistributed, this has the effect of redistributing the stresses amongst adjacent Gauss points ensuring that the yield criteria is satisfied, see Section 2.6. The elastic analyses and peak stress envelope results presented earlier are certainly under-estimating the number of integration points which have reached a peak stress caused by the pre-stressing wires failing. A non linear analysis could result in different trends in tensile and compressive behaviour being seen because in such analyses, stresses will reduce and rotation of the principal stress directions will take place.



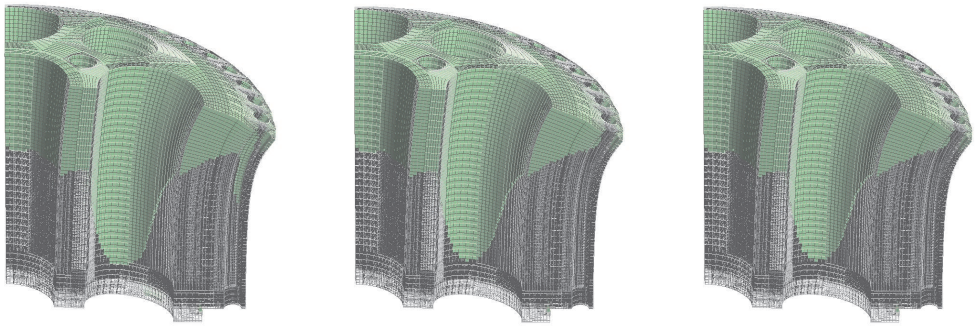
(a) MPenv (b) MPenv (c) TCenv

Figure 5.12: Deformation mesh plots (scale factor 200) showing the elements where the stresses lie outside the peak stress envelopes: mesh 8, load case 1



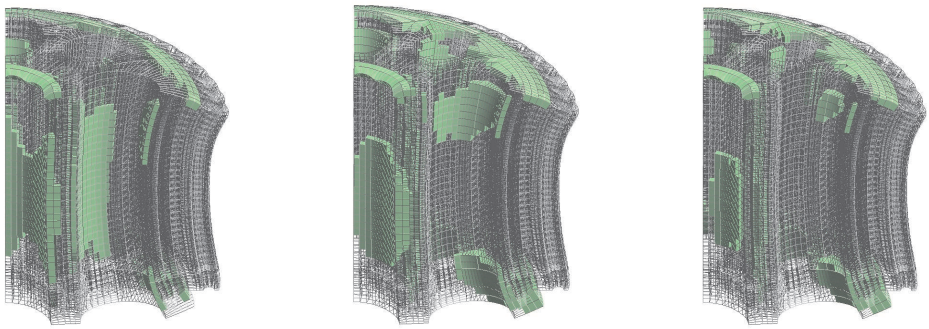
(a) Senv (b) MPenv (c) TCenv

Figure 5.13: Deformation mesh plots (scale factor 200) showing the elements where the stresses lie outside the peak stress envelopes: mesh 8, load case 2



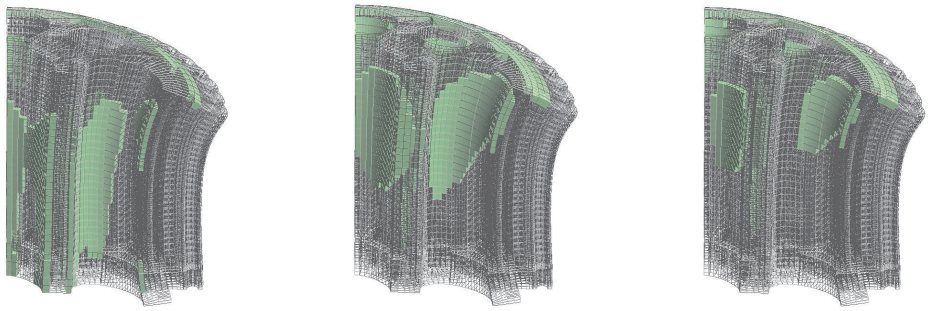
(a) Senv (b) MPenv (c) TCenv

Figure 5.14: Deformation mesh plots (scale factor 200) showing the elements where the stresses lie outside the peak stress envelopes: mesh 8, load case 3



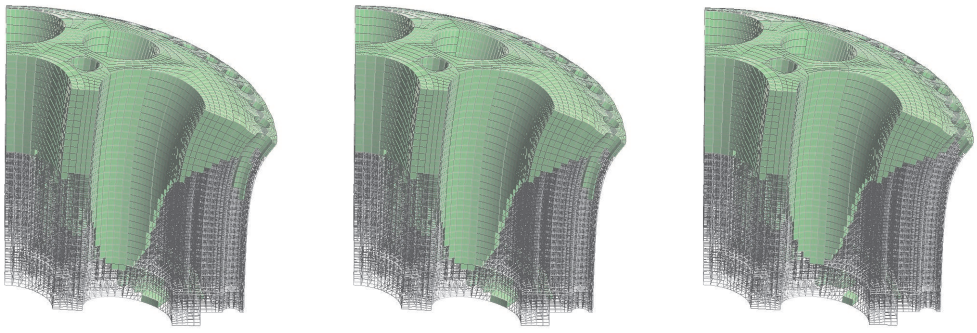
(a) Senv (b) MPenv (c) TCenv

Figure 5.15: Deformation mesh plots (scale factor 200) showing the elements where the stresses lie outside the peak stress envelopes: mesh 20, load case 1



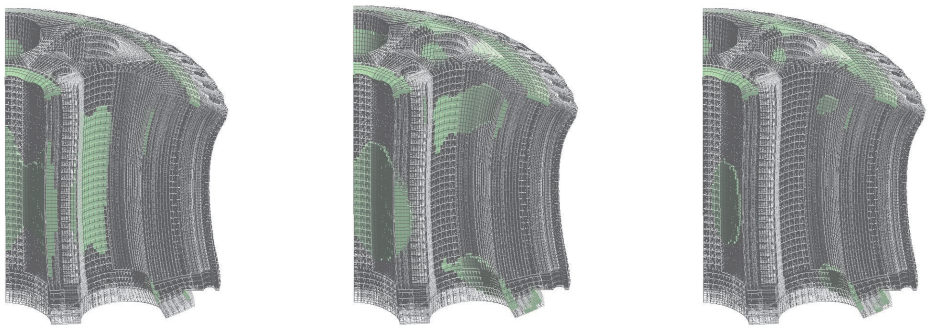
(a) Senv (b) MPenv (c) TCenv

Figure 5.16: Deformation mesh plots (scale factor 200) showing the elements where the stresses lie outside the peak stress envelopes: mesh 20, load case 2



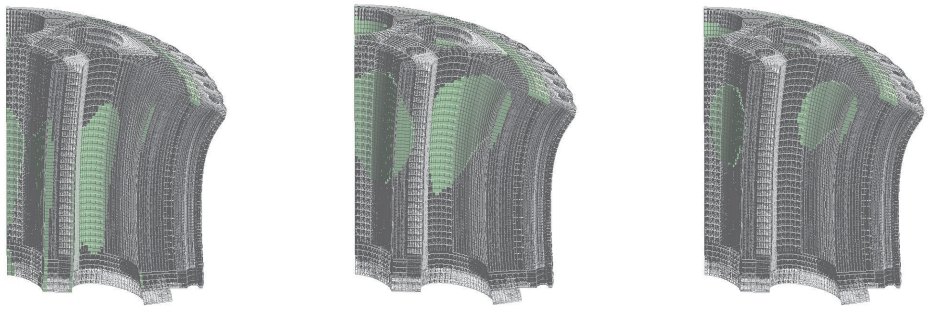
(a) Senv (b) MPenv (c) TCenv

Figure 5.17: Deformation mesh plots (scale factor 200) showing the elements where the stresses lie outside the peak stress envelopes: mesh 20, load case 3



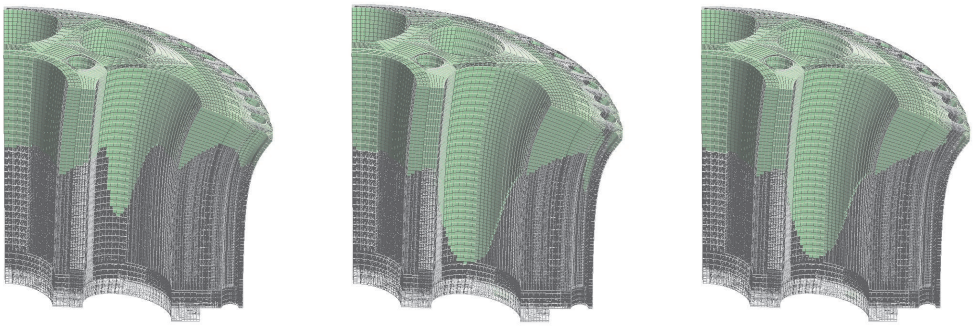
(a) Senv (b) MPenv (c) TCenv

Figure 5.18: Deformation mesh plots (scale factor 200) showing the elements where the stresses lie outside the peak stress envelopes: mesh 88, load case 1



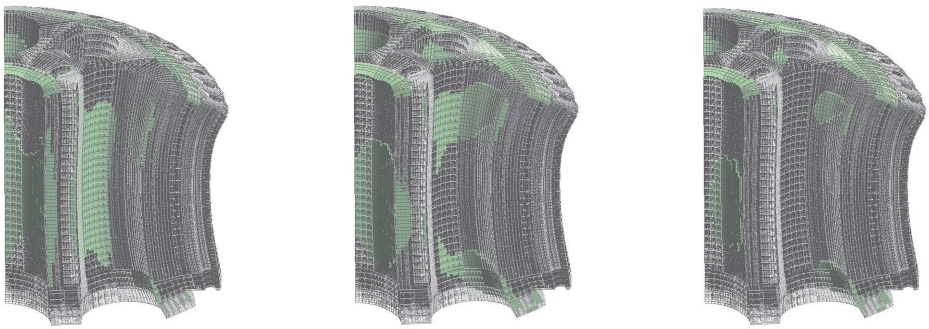
(a) Senv (b) MPenv (c) TCenv

Figure 5.19: Deformation mesh plots (scale factor 200) showing the elements where the stresses lie outside the peak stress envelopes: mesh 88, load case 2



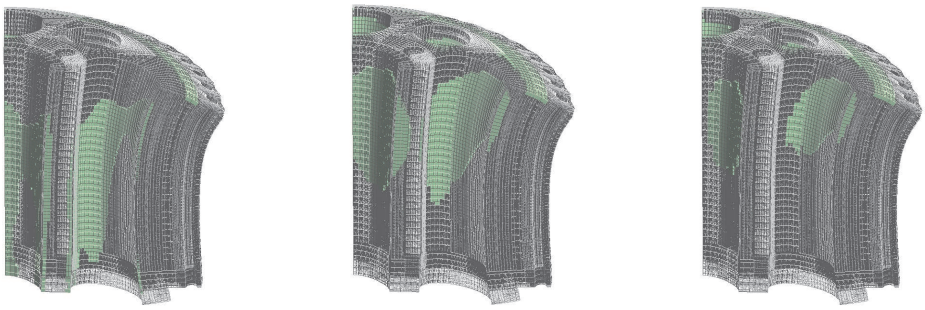
(a) Senv (b) MPenv (c) TCenv

Figure 5.20: Deformation mesh plots (scale factor 200) showing the elements where the stresses lie outside the peak stress envelopes: mesh 88, load case 3



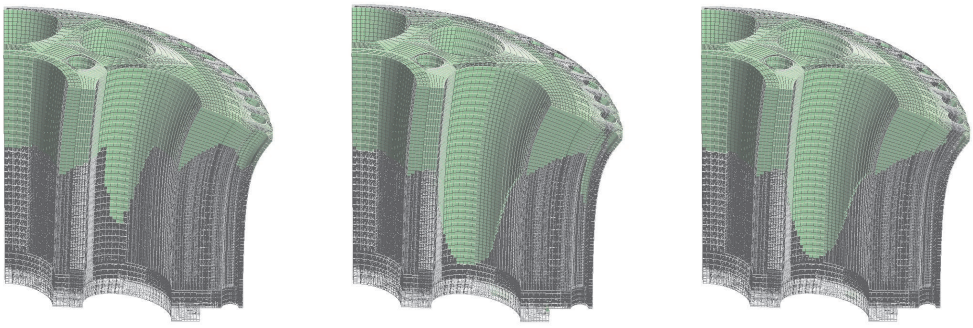
(a) Senv (b) MPenv (c) TCenv

Figure 5.21: Deformation mesh plots (scale factor 200) showing the elements where the stresses lie outside the peak stress envelopes: mesh 820, load case 1



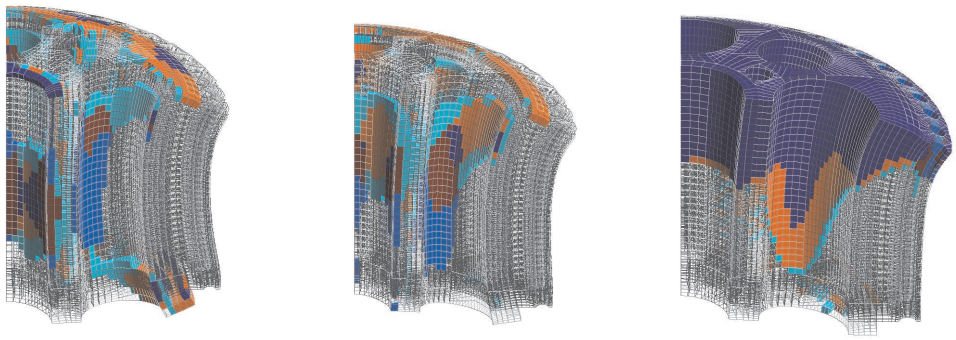
(a) Senv (b) MPenv (c) TCenv

Figure 5.22: Deformation mesh plots (scale factor 200) showing the elements where the stresses lie outside the peak stress envelopes: mesh 820, load case 2



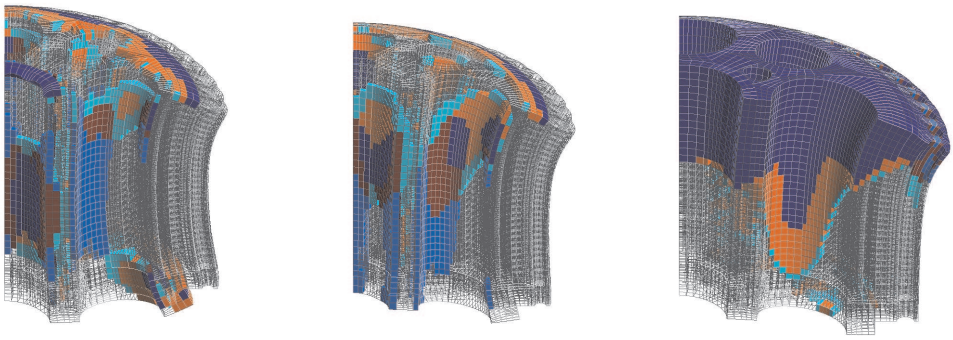
(a) Senv (b) MPenv (c) TCenv

Figure 5.23: Deformation mesh plots (scale factor 200) showing the elements where the stresses lie outside the peak stress envelopes: mesh 820, load case 3



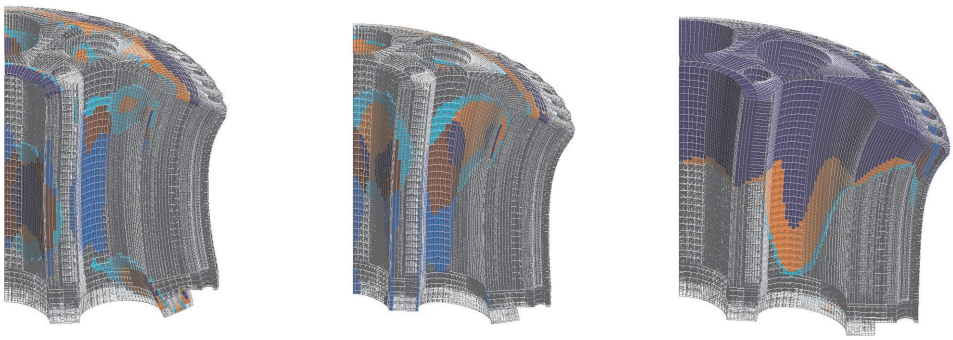
(a) Load case 1 (b) Load case 2 (c) Load case 3

Figure 5.24: Deformation mesh plots (scale factor 200) showing the elements where the stresses lie outside the peak stress envelopes: mesh 8, load cases 1-3 (see the colour code given on Figure 5.28)



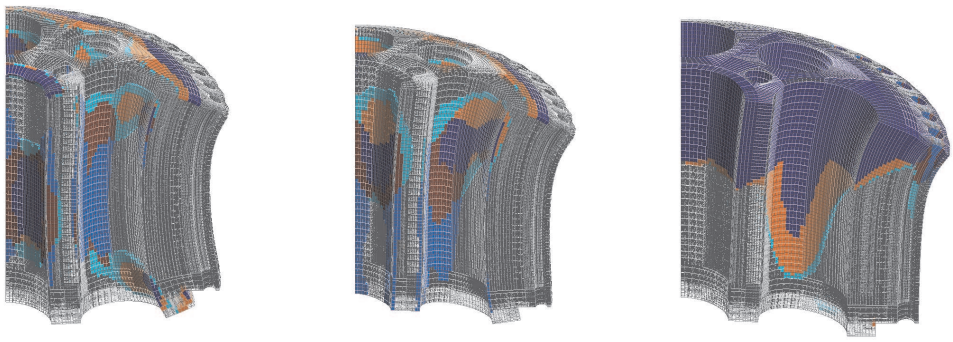
(a) Load case 1 (b) Load case 2 (c) Load case 3

Figure 5.25: Deformation mesh plots (scale factor 200) showing the elements where the stresses lie outside the peak stress envelopes: mesh 20, load cases 1-3 (see the colour code given on Figure 5.28)



(a) Load case 1 (b) Load case 2 (c) Load case 3

Figure 5.26: Deformation mesh plots (scale factor 200) showing the elements where the stresses lie outside the peak stress envelopes: mesh 88, load cases 1-3 (see the colour code given on Figure 5.28)



(a) Load case 1 (b) Load case 2 (c) Load case 3

Figure 5.27: Deformation mesh plots (scale factor 200) showing the elements where the stresses lie outside the peak stress envelopes: mesh 820, load cases 1-3 (see the colour code given on Figure 5.28)

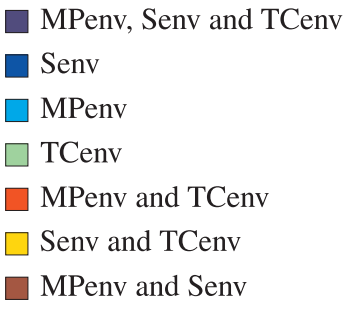
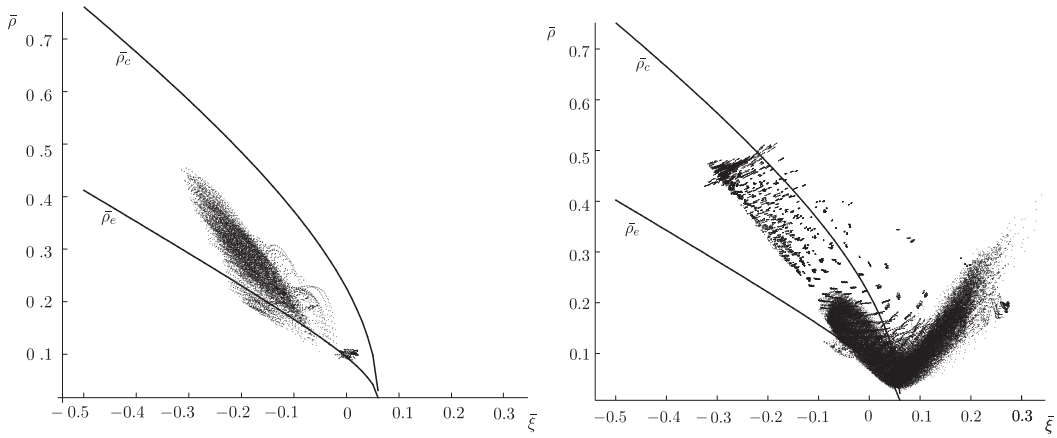
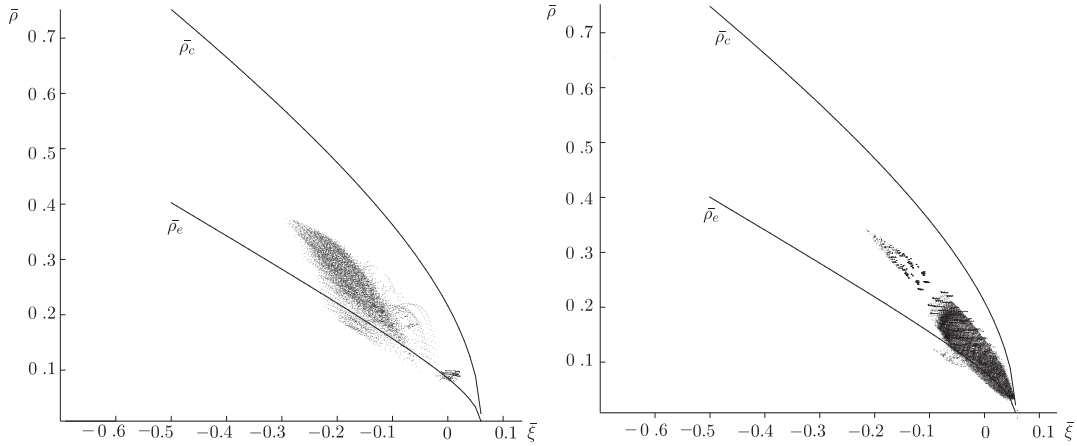


Figure 5.28: Peak stress envelope indicators used to identify where the stresses exceed the strength limit



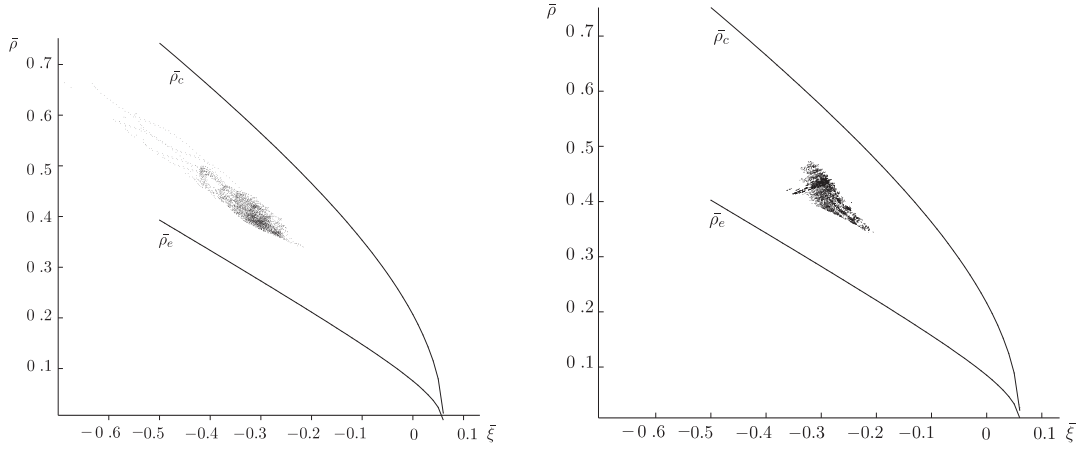
(a) Load case 2 (b) Load case 3

Figure 5.29: Meridional section for Senv showing location of stress states which lie outside MPenv



(a) Load case 2 (b) Load case 3

Figure 5.30: Meridional section for Senv showing location of stress states which lie outside MPenv and inside Senv



(a) Load case 2 (b) Load case 3

Figure 5.31: Meridional section for Senv showing location of stress states which lie inside MPenv and outside Senv

5.6.4 Principal stress plots

The BCU was designed to operate under compressive stresses. Figures 5.32, 5.33 and 5.34 show the tensile principal stress components for Mesh 8. It is clear that for load case 3 (Figure 5.32) the number of stress states with a tensile component is significant. The number of plotted lines has increased from the few visible along the underside, bolts and inner penetration of the BCU under load case 1 (Figure 5.32)). This is seen in particular across the top surface of the BCU under load case 3 (Figure 5.34) where the structure curves due to the under pressure and bolt loads. The magnitudes of these tensile stresses are largest in this area. The tensile stresses inside the penetrations increase as they approach the top of the BCU. Coloured principal stress plots for load cases 2 and 3 have been plotted looking at three horizontal cross sections through the BCU structure for a single level of Gauss points(that is planes parallel to the $x-z$ axis: Cuts A,B and C). These cuts are shown in (Figures 5.35, 5.36 and 5.37). Tensile stress components have been plotted in red and compressive stress components in blue. The orientation of the vectors indicate the direction of the principal stress axes and the length of the lines indicates the magnitude of the principal stresses. In this way, the flow of stress through the BCU can be determined. It is clear when considering load cases 2 and 3 that the BCU moves from a state where many of the stress components are compressive to a state where most of the stress components are tensile when the pre-stress is removed. The tensile stresses are largest on the top surface of the BCU. This is visible in Figure 5.35 for load case 3 where the tensile stresses coloured in blue are much larger than for the other two planes under this load case shown in Figures 5.36 and 5.37 (the 'blueness' of the plots is much stronger). Under load case 2 there are small areas where tensile principal stress components exist, most notably on the approach to the bolts on the top surface (cut A) , see Figure 5.35(a). The size of the tensile area is significantly larger and the magnitude of the tensile principal stresses increased when the pre-stressing is removed in load case 3, see Figure 5.35(b). This supports the findings of Section 5.6.2 that suggested that under the extreme case of load case 3 the nuclear reactor is no longer safe to operate due to concrete's low strength in tension.

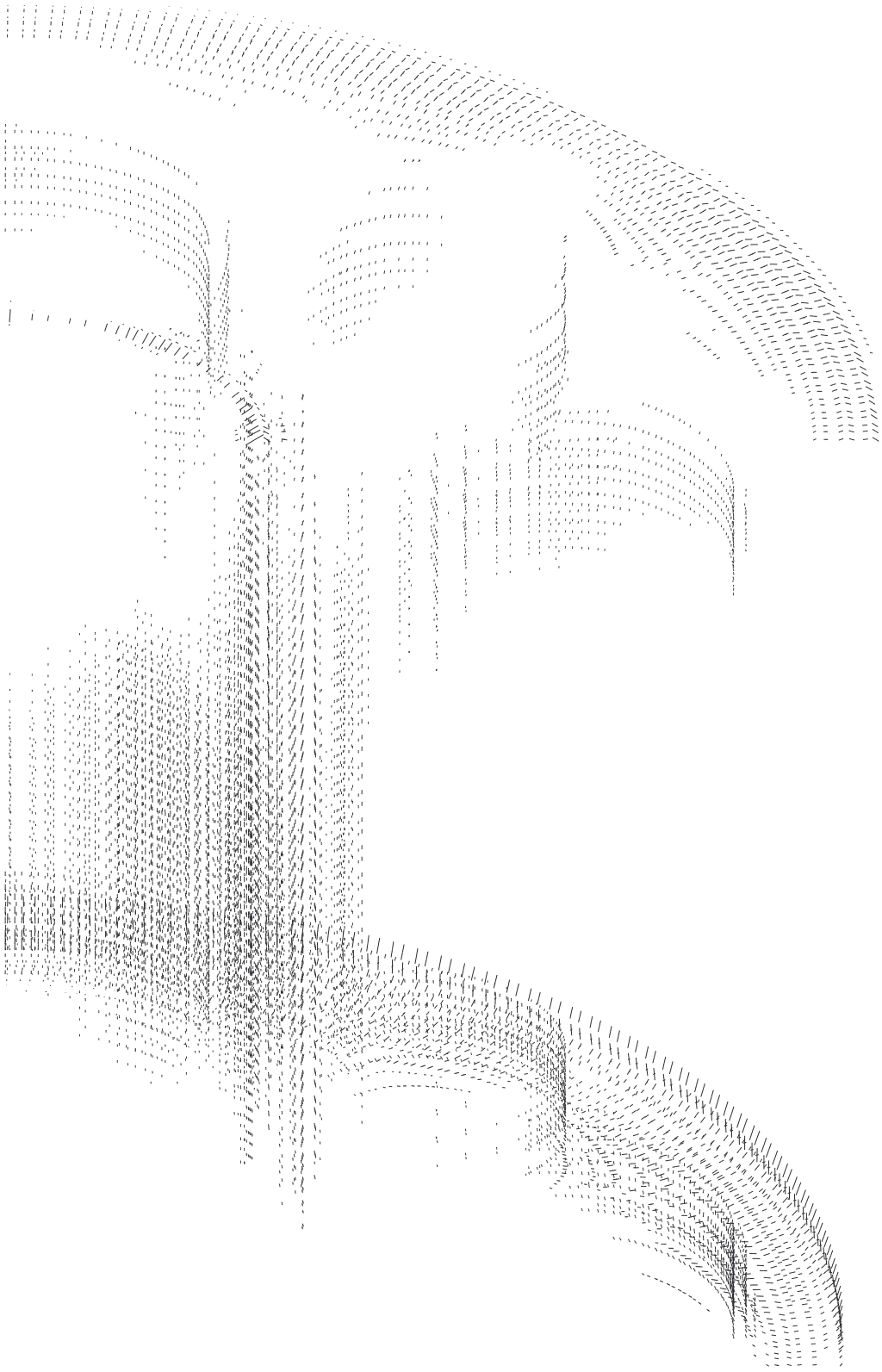


Figure 5.32: Tensile principal stress vectors located outside MPenv - load case 1

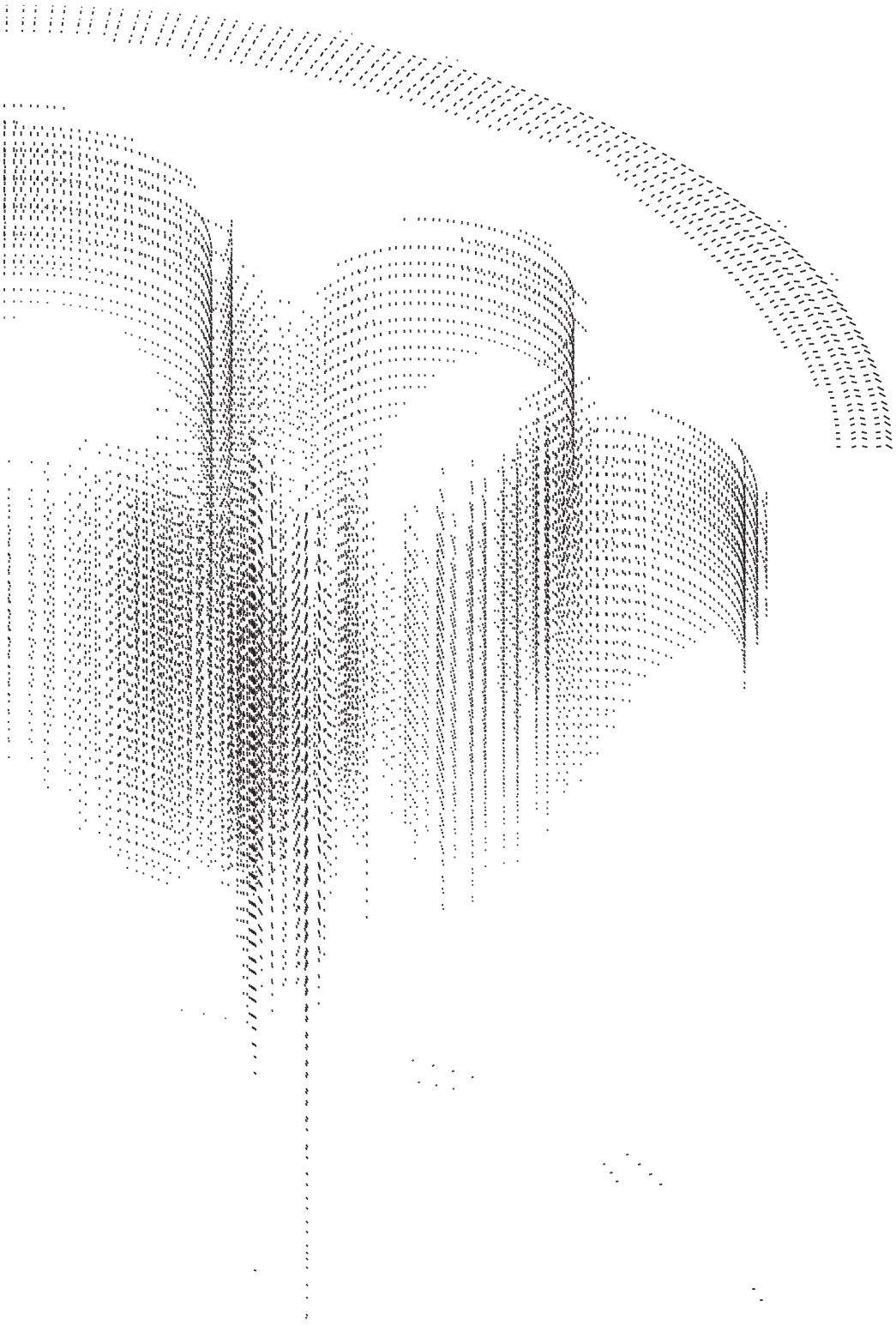


Figure 5.33: Tensile principal stress vectors located outside MPenv - load case 2

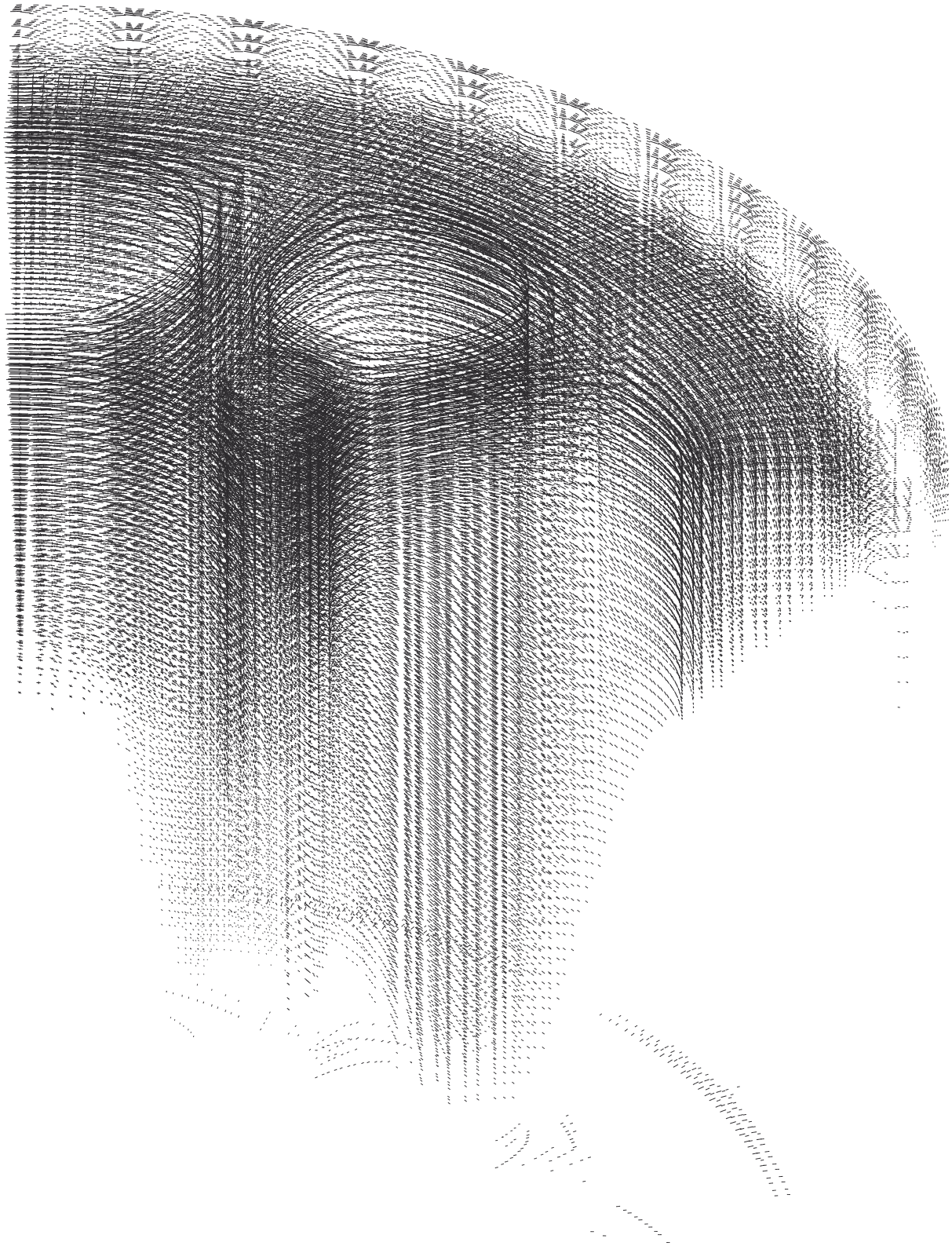
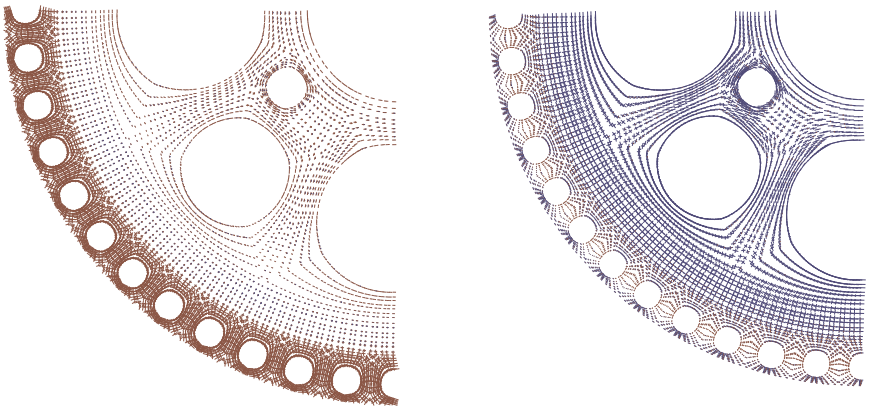
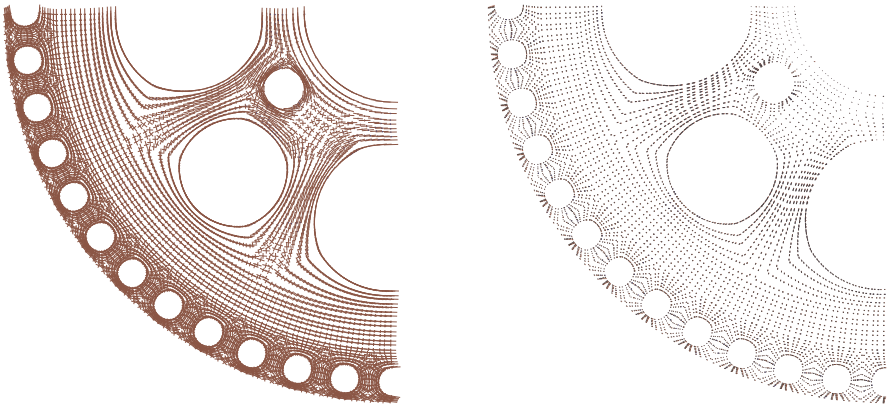


Figure 5.34: Tensile principal stress vectors located outside MPenv - load case 3



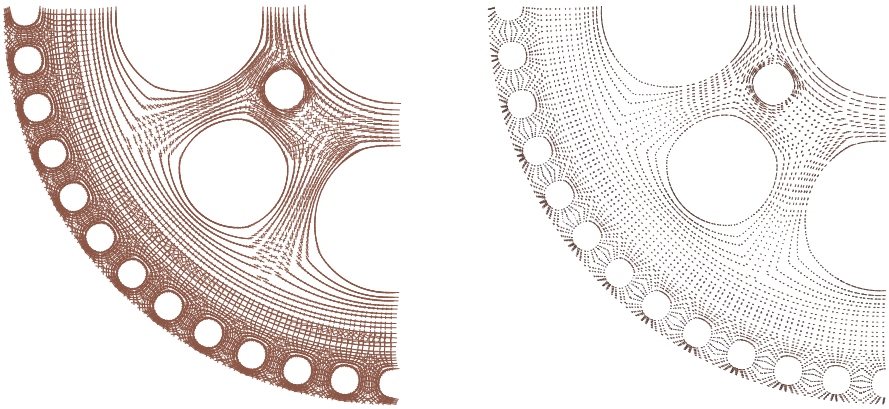
(a) Load case 2 (b) Load case 3

Figure 5.35: Cross section A: tensile and compressive principal stress vectors near the top surface of BCU (at depth $y = 1697$)



(a) Load case 2 (b) Load case 3

Figure 5.36: Cross section B: tensile and compressive principal stress vectors in the middle of the BCU (at depth $y = 866$)



(a) Load case 2 (b) Load case 3

Figure 5.37: Cross section C: tensile and compressive principal stress vectors near the bottom of the BCU (at depth $y = 111$)

5.6.5 Error analysis

For each of the four meshes and for the three load cases, error analyses has been performed using the Z^2 error estimator, see Section 4.4. The error in energy norm $|e|$ is calculated for each element. The maximum error in energy norm, $|e|_{max}$ and the mean error in energy norm $|\overline{e}|$ for the whole mesh have been recorded in Table 5.9. The element at which the $|e|_{max}$ occurs has also been recorded in square brackets after $|e|_{max}$. Figure 5.38 highlights the elements where the maximum error in energy norm occurs for each BCU mesh, and the element numbers have also been displayed in square brackets. For load cases 1 and 2 the maximum error is generally seen in an element close to the step on the underside of the BCU. The exceptions are for Mesh 8 load case 1 where the maximum error occurs in the smallest penetration near the bottom of the BCU and for Mesh 820, load case 1 where the maximum error occurs on the top surface in the inner penetration where the weight of the boiler is acting. For load case 3, the maximum error for all four meshes is seen on the top surface close to the bolts where large distortion occurs. If local refinement, see Section 4.3.3, was to be carried out it should take place around the penetrations (especially around the elements where the weight of the boiler is loaded at the top of the inner penetration), near the bolts and around the step on the underside of the BCU. From the displacement and peak stress envelope results shown in this chapter, although the results for the four meshes are not identical the main trends are captured by the coarsest mesh (Mesh 8). On average this took less than a minute to run (when using the optimum linear solver and number of processors see Table 5.4) in comparison with 262 minutes for mesh 820.

Load Case	1		2		3	
Mesh	$\|\overline{e}\|$	$\|e\|_{max}$	$\|\overline{e}\|$	$\|e\|_{max}$	$\|\overline{e}\|$	$\|e\|_{max}$
8	2.8577	28.7821 [36993]	2.685	15.4395 [29539]	0.8918	9.7614 [16011]
20	0.9298	26.8783 [36995]	0.8223	9.2894 [14821]	0.2418	8.3074 [16487]
88	0.4432	17.9764 [295941]	0.412	4.7804 [118561]	0.1527	3.5605 [128085]
820	0.1054	17.7006 [236937]	0.0898	4.0005 [118561]	0.0252	2.8992 [320244]

Table 5.9: BCU SPR error analysis results

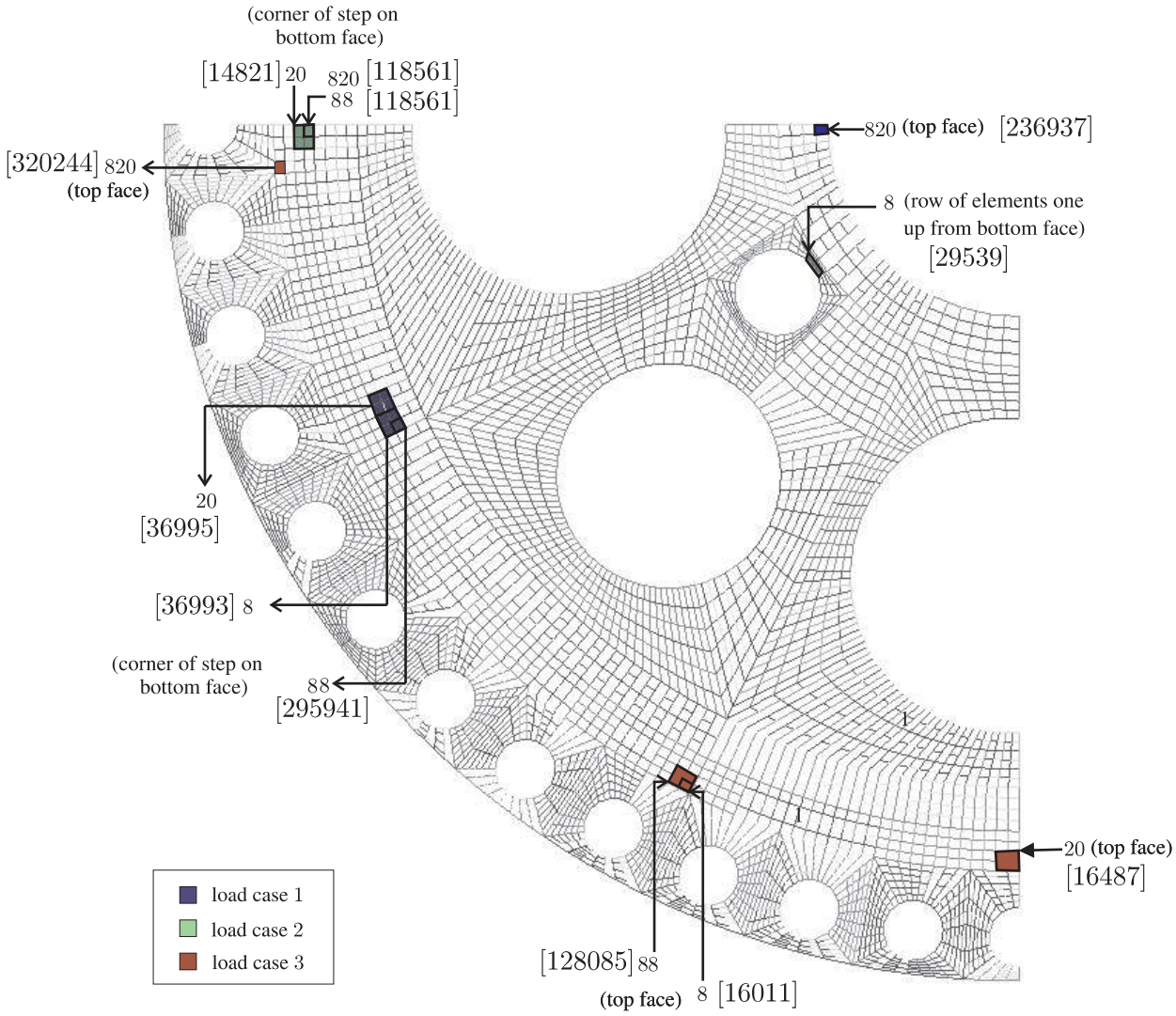


Figure 5.38: Location of the element where the maximum error in energy norm occurs for each BCU mesh

Chapter 6

Conclusions and recommendations for further work

Here the original contributions made by this project will be summarised. The key findings are highlighted and suggestions made for further work. Four original contributions have been made.

1. An automatic subdivision program has been written in Fortran 90 (`mesh_refine.f90`). This program can take any 8 or 20-noded hexahedral mesh and refine the whole domain further, by either h-refinement (increasing the number of nodes per element from 8 to 20) or by p-refinement (subdividing each 20-noded element into eight 8-noded elements). This program has been successfully run, refining a 40201 element 8-noded hexahedral mesh (Mesh 8) three times to produce three new meshes: a 40201 element 20-noded hexahedral mesh (Mesh 20), a 120603 element 8-noded hexahedral mesh (Mesh 88) and finally a 120603 element 20-noded hexahedral mesh (Mesh 820). The full program including subroutines for both refinement algorithms, reading input mesh data and outputting mesh data for both `ParaFEM` and `Gmsh` is approximately 700 lines of code. It takes approximately 10 minutes to convert Mesh 8 to Mesh 20, 100 minutes to convert Mesh 20 to Mesh 88 and 200 minutes to convert Mesh 88 to Mesh 820 when the program is run on the linux service at Durham University (*Vega*).

2. `MUMPS`, a parallel direct linear solver, has been successfully introduced into `ParaFEM`. In order to run this parallel direct solver, four new Fortran sub-routines were written (`calc_ke_red`, `calc_urdof`, `calc_eltvar` and `calc_NZ`) and organised within a new module `MUMPS.mod`.

3. The Z^2 error estimator has been coded for a 3D mesh (in total 453 lines of Fortran code incorporated into `ParaFEM`'s `p121.f90`), and a new strategy has been developed to handle hexahedral elements at mesh boundaries. This 3D Z^2 error estimator has been used to carry out an error analysis on the BCU meshes. Error analysis for Mesh 8 takes under 30 minutes to complete, while error analysis for Mesh 820 takes approximately 10 hours.

4. A simple multi-planar peak stress envelope (MPenv) has been devised to analyse the FEA results. This criterion has been compared with a tension cut off peak stress envelope (TCenv) and a smooth

C2 yield surface (Senv).

As described in Section 3.2.2, `MUMPS` was successfully incorporated into `ParaFEM` and the correct displacements were outputted from the FEA. Speed up was achieved, however parallel benefits were very limited when running `p121.f90` on multi-cores on *Hamilton* and problems were experienced when running the analysis on the largest mesh (Mesh 820 containing 1385616 nodes). When the original iterative solver was used, speed up was seen when using up to sixteen cores for the larger meshes, but when using the `MUMPS` solver, although the problem was generally solved quicker, speed up was only seen when using up to four cores.

When analysing the BCU, the results confirm that the pre-stressing wire windings are necessary to retain the BCU concrete in a state of multi-axial compression. When the pre-stressing is removed (load case 3) a large area of the BCU experiences tensile stresses. These areas of tension indicate a structural problem because concrete has very little strength in tension. The peak stress envelope analyses also show how under load case 3 many more of the BCU's Gauss point stress states lie outside the peak stress envelopes. Both these findings suggest that if the pre-stressing wire windings snap or deteriorate, the structural integrity of the BCU is lost and the nuclear reactor is no longer safe to operate.

Automatic mesh refinement was successfully implemented. The FEA results for the BCU show that although progressive refinement reduces the error in the FEA, Mesh 8, the coarsest BCU mesh, provides an adequate mesh to capture the trends in displacement and structural condition.

By comparing peak stress envelopes it became clear the the TCenv was an insufficient model for determining the failure of concrete due to concrete's multi-axial behaviour. Both MPenv and Senv picked up on stress states outside the peak stress envelope which were not identified by TCenv. However Senv and MPenv provide different models to represent concrete's limiting multi-axial strength. The differences between the models resulted in different areas of the BCU being highlighted as having stress states outside the envelope, with MPenv being the most conservative model and identifying the most elements outside the envelope. Although peak stress envelopes such as Senv and MPenv attempt to model this behaviour, they have not been satisfactorily validated due to lack of experimental data, particularly in the multi-axial tension-compression regions. The results presented here highlight how using different models can result in different conclusions being drawn on the structural condition of different parts of the mesh under multi-axial loading conditions. It is therefore important to consider such loading conditions and take care when using one peak stress envelope model in isolation.

The development of a parallel FE code for the elastic analysis of the BCU has provided an interesting initial study, allowing the structural condition of the BCU to be assessed and the suitability of peak stress envelopes to be discussed. To investigate this problem further and to provide a more realistic analysis of the BCU's behaviour the code should be extended to allow non-linear analysis to be carried out, see Section 2.6. It would be interesting to see exactly how the trends (about the structural condition of the BCU) suggested by the elastic analysis alter when carrying out a non-linear analysis.

Appendix A

ParaFEM Input Files

Example input files are provided in this appendix. The problem presented is a unit cube made up of eight, 8-noded hexahedral elements subjected to a uniform pressure $p = 1$ and a single point load $f = 1$ (Figure A.1). The boundary conditions for this problem are roller boundary conditions on two corner faces and on the bottom face, as used in the unit cube problems of Chapter 4.

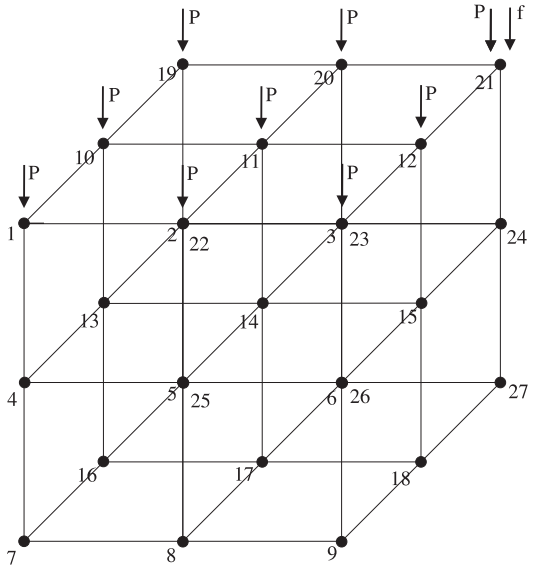


Figure A.1: Example problem

A.1 p.121.dat

p121.dat is the control data file. It provides data about the problem such as the number of elements, nodes and restrained degrees of freedom.

Variable Name	Type	Purpose
element	character	the element type: hexahedron or tetrahedron
mesh	integer	element node ordering scheme: 1-Smith and Griffiths or 2-Abaqus
nels	integer	number of elements in mesh
nn	integer	number of nodes in mesh
nr	integer	number of integration points
nip	integer	number of nodes per element
nod	integer	number of nodes with fixed displacements
loaded_nodes	integer	number of nodes with externally applied loads
e	real	Young's modulus
v	real	Poisson's ratio
tol	real	convergence tolerance for PCG
limit	integer	iteration ceiling for PCG
pressure_faces	integer	number of element surfaces with applied pressure

Table A.1: Variables listed in p121.dat

```
element
mesh
nels nn nr nip nod loaded_nodes
e v tol limit pressure_faces
```

```
hexahedron
2
8 27 19 8 8 1
0.100E+04 0.2 0.100E-08 100000. 4
```

Figure A.2: Example p121.dat file

A.2 p.121.d

p121.d provides data on the nodal coordinates and element topology.

Variable Name	Type	Purpose
*THREE_DIMENSIONAL	character	a keyword describing the model as three dimensional
*NODES	character	a keyword marking the start of a list of nodes and their coordinates
*ELEMENTS	character	a keyword marking the strat of a list of element data
nodeID	integer	a unique number that identifies the node. ParaFEM assumes sequential numbering from 1 to nn
x-co-ordinate	real	the x-co-ordinate of the node
y-co-ordinate	real	the y-co-ordinate of the node
z-co-ordinate	real	the z-co-ordinate of the node
elementID	integer	a unique number that identifies the element. ParaFEM assumes sequential numbering from 1 to nels
ndim	integer	the number of dimensions. ParaFEM only supports 3D elements
nod	integer	the number of nodes in the element. ParaFEM only supports the values 4, 8, 10 and 20
num	integer list	element topology listed according to Smith and Griffiths or Abaqus node numbering
materialID	integer	a number that identifies which material properties to select for the element

Table A.2: Variable listed in p121.d

*THREE_DIMENSIONAL
*NODES
nodeID x-coordinate y-coordinate z-coordinate
*ELEMENTS
elementID ndim nod type num materialID

```
*THREE_DIMENSIONAL
*NODES
1       0.00000000E+00 0.00000000E+00 0.00000000E+00
2       0.50000000E+00 0.00000000E+00 0.00000000E+00
3       0.10000000E+01 0.00000000E+00 0.00000000E+00
4       0.00000000E+00 0.00000000E+00 -0.50000000E+00
5       0.50000000E+00 0.00000000E+00 -0.50000000E+00
6       0.10000000E+01 0.00000000E+00 -0.50000000E+00
7       0.00000000E+00 0.00000000E+00 -0.10000000E+01
8       0.50000000E+00 0.00000000E+00 -0.10000000E+01
9       0.10000000E+01 0.00000000E+00 -0.10000000E+01
10      0.00000000E+00 0.50000000E+00 0.00000000E+00
11      0.50000000E+00 0.50000000E+00 0.00000000E+00
12      0.10000000E+01 0.50000000E+00 0.00000000E+00
13      0.00000000E+00 0.50000000E+00 -0.50000000E+00
14      0.50000000E+00 0.50000000E+00 -0.50000000E+00
15      0.10000000E+01 0.50000000E+00 -0.50000000E+00
16      0.00000000E+00 0.50000000E+00 -0.10000000E+01
17      0.50000000E+00 0.50000000E+00 -0.10000000E+01
18      0.10000000E+01 0.50000000E+00 -0.10000000E+01
19      0.00000000E+00 0.10000000E+01 0.00000000E+00
20      0.50000000E+00 0.10000000E+01 0.00000000E+00
21      0.10000000E+01 0.10000000E+01 0.00000000E+00
22      0.00000000E+00 0.10000000E+01 -0.50000000E+00
23      0.50000000E+00 0.10000000E+01 -0.50000000E+00
24      0.10000000E+01 0.10000000E+01 -0.50000000E+00
25      0.00000000E+00 0.10000000E+01 -0.10000000E+01
26      0.50000000E+00 0.10000000E+01 -0.10000000E+01
27      0.10000000E+01 0.10000000E+01 -0.10000000E+01
*ELEMENTS
1 3 8 1   4 5 14 13 1 2 11 10   1
2 3 8 1   5 6 15 14 2 3 12 11   1
3 3 8 1   7 8 17 16 4 5 14 13   1
4 3 8 1   8 9 18 17 5 6 15 14   1
5 3 8 1   13 14 23 22 10 11 20 19   1
6 3 8 1   14 15 24 23 11 12 21 20   1
7 3 8 1   16 17 26 25 13 14 23 22   1
8 3 8 1   17 18 27 26 14 15 24 23   1
```

Figure A.3: Example `p121.d` file

A.3 p.121.bnd

p121.bnd provides data on the boundary conditions.

Variable Name	Type	Purpose
nodeID	integer	a unique number that identifies the node, nodes were all the restraints are free need not be included
restraint_x	integer	restraint in x direction, the convention is that 0=restrained and 1=free
restraint_y	integer	restraint in y direction, the convention is that 0=restrained and 1=free
restraint_z	integer	restraint in z direction, the convention is that 0=restrained and 1=free
nr	integer	number of restrained node

Table A.3: Variables listed in p121.bnd

nodeID-1 restraint_x restraint_y restraint_z
nodeID-2 restraint_x restraint_y restraint_z
nodeID-3 restraint_x restraint_y restraint_z
...
nodeID-nr restraint_x restraint_y restraint_z

```
1       0  0  1
2       1  0  1
3       1  0  1
4       0  0  1
5       1  0  1
6       1  0  1
7       0  0  0
8       1  0  0
9       1  0  0
10      0  1  1
13      0  1  1
16      0  1  0
17      1  1  0
18      1  1  0
19      0  1  1
22      0  1  1
25      0  1  0
26      1  1  0
27      1  1  0
```

Figure A.4: Example p121.bnd file

A.4 p.121.lds

p121.lds provides data on the externally applied nodal point loads.

Variable Name	Type	Purpose
nodeID	integer	a unique number that identifies the loaded node, nodes which do not have applied loads need not be included
value_x	real	value of load applied in x direction
value_y	real	value of load applied in y direction
value_z	real	value of load applied in z direction
loaded_nodes	integer	the number of loaded nodes in the model

Table A.4: Variables listed in p121.lds

```
nodeID-1 value_x value_y value_z
nodeID-2 value_x value_y value_z
nodeID-3 value_x value_y value_z
...
nodeID-loaded_nodes value_x value_y value_z
```

```
20 0 0 1.0
```

Figure A.5: Example p121.lds file

A.5 p.121.prs

p121.prs provides data on the pressures applied to element faces, for 8 and 20-noded hexahedral elements only.

Variable Name	Type	Purpose
elementID	integer	a unique number that identifies the element
prs_face_num	integer list	list of node numbers to identify element face, list in a clockwise order when looking from an external position onto the face
value_a	real	shear force acting across element face in direction from surface node 1 to 2
value_b	real	pressure acting perpendicular to element face
value_c	real	shear force acting across element face in direction from surface node 2 to 3 (8-noded hexahedron) and from surface node 3 to 4 (20-noded hexahedron)
snod	integer	number of surface nodes, 4 for 8-noded hexahedron, 8 for 20-noded hexahedron

Table A.5: Variables listed in p121.prs

elementID
pressure_face_num
value_a1 value_b1 value_c1 ... value_a_snod value_b_snod value_c_snod

```
1
1 10 11 2
0 1.0 0 0 0 1.0 0 0 1.0 0 0 1.0 0
2
2 11 12 3
0 1.0 0 0 0 1.0 0 0 1.0 0 0 1.0 0
5
10 19 20 11
0 1.0 0 0 0 1.0 0 0 1.0 0 0 1.0 0
6
11 20 21 12
0 1.0 0 0 0 1.0 0 0 1.0 0 0 1.0 0
```

Figure A.6: Example p121.prs file

Appendix B

`make` File Example

This appendix provides the `make` file used to compile the main program `p121.f90` with all the module files containing the subroutines. Comments are given to explain the purpose of each part of the file.

```
#Lorna van Griethuysen 21/10/10
#Makefile for PARAFEM

F90C = mpif90
EXE  = p121_exe

OBJ = p121.o precision.o global_variables.o mp_interface.o input.o
      output.o maths.o gather_scatter.o pcg.o partition.o steering.o
      elements.o loading.o timing.o mumps.o gmsh.o spr.o

MOD = elements.mod gather_scatter.mod global_variables.mod input.mod
      loading.mod maths.mod mp_interface.mod output.mod partition.mod
      pcg.mod  precision.mod steering.mod timing.mod mumps.mod gmsh.mod
      spr.mod

topdir = ..
libdir = $(topdir)/lib

include $(topdir)/Makefile.inc
LIBMUMPS_COMMON = $(libdir)/libmumps_common$(PLAT).a
LIBDMUMPS = $(libdir)/libdmumps$(PLAT).a $(LIBMUMPS_COMMON)

.KEEP_STATE:

#Linking all object files to produce an executable
$(EXE): $(LIBDMUMPS) $(OBJ)
$(F90C) -o $(EXE) $(OBJ) $(LIBDMUMPS) $(LIB) $(LIBBLAS)
           $(LIBOTHERS)$(LORDERINGS) $(LPORD)

#Compiling main program
p121.o: p121.f90 $(MOD)
$(F90C) $(OPTF) $(INC) -I. -I$(topdir)/include  -c p121.f90

#Compiling each module of subroutines
pcg.mod: pcg.f90 precision.mod maths.mod gather_scatter.mod
$(F90C) -c pcg.f90

input.mod: input.f90 precision.mod mp_interface.mod loading.mod
$(F90C) -c input.f90

loading.mod: loading.f90 precision.mod gather_scatter.mod
             maths.mod steering.mod
$(F90C) -c loading.f90

spr.mod: spr.f90 maths.f90 precision.mod mp_interface.mod
         mumps.mod
  $(F90C) -c spr.f90

maths.mod: maths.f90 precision.mod mp_interface.mod
$(F90C) -c maths.f90

gather_scatter.mod: gather_scatter.f90 mp_interface.mod
                    global_variables.mod precision.mod
$(F90C) -c gather_scatter.f90
```

assign shorthand for: compiler, name of executable, list of object files and list of module files

assign shorthand for: directory locations

provide location of supplementary `MUMPS` `make` file and `MUMPS` libraries

checks and updates compile options within the `make` file every time the file is run

main program linked with all object files to produce and executable

main program compiled depending on all module files

each module file is compiled depending on other module files listed as dependancies

Figure B.1: Example `make` file (a)

```
output.mod: output.f90 precision.mod mp_interface.mod
$(F90C) -c output.f90

gmsh.mod: gmsh.f90 precision.mod mp_interface.mod output.mod
$(F90C) -c gmsh.f90

mumps.mod:  mumps.f90 precision.mod mp_interface.mod
$(F90C) -c mumps.f90

mp_interface.mod: mp_interface.f90 global_variables.mod
                  precision.mod
$(F90C) -c mp_interface.f90

elements.mod: elements.f90 precision.mod
$(F90C) -c elements.f90

steering.mod: steering.f90 precision.mod
$(F90C) -c steering.f90

partition.mod: partition.f90 precision.mod
$(F90C) -c partition.f90

global_variables.mod: global_variables.f90 precision.mod
$(F90C) -c global_variables.f90

timing.mod: timing.f90 precision.mod
$(F90C) -c timing.f90

precision.mod: precision.f90
$(F90C) -c precision.f90

$(libdir)/libdmumps$(PLAT).a:
@echo 'Error: you should build the library' $@ 'first'
exit -1

clean:
-rm -f *.mod *.o
```

each module file is compiled depending on other module files listed as dependancies

error message if `MUMPS` library cannot be found

`make clean` command removes all existing object and module files

Figure B.2: Example `make` file continued (b)

Appendix C

Code Appendix

C.1 ParaFEM - with MUMPS solver

The code presented here is the adapted ParaFEM `p121` program. The `MUMPS` solver has been adapted. `MUMPS` is called at the start of the program and the appropriate parameters set to allow the program to be run of Hamilton and to analysis the BCU. All ParaFEM modules have been left untouched. The SPR error estimator has also been added and the code is visible at the end of the program.

```
PROGRAM p121
!-------------------------------------------------------------------
!      Program 12.1 three dimensional analysis of an elastic solid
!      Adapted by Lorna van Griethuysen
!      MUMP solver added - PCCG solver removed
!      SPR error estimator added
!-------------------------------------------------------------------


  USE precision     ; USE global_variables ; USE mp_interface
  USE input         ; USE output           ; USE loading
  USE timing        ; USE maths            ; USE gather_scatter
  USE partition     ; USE elements         ; USE steering
  USE pcg           ; USE mumps

  IMPLICIT NONE

  INCLUDE "dmumps_struc.h"
  TYPE (DMUMPS_STRUC)id

!-------------------------------------------------------------------
```

```
! 1. Declare variables used in the main program
!-----------------------------------------------------------------

 ! neq,ntot are now global variables - not declared

  INTEGER,PARAMETER     :: nodof=3,ndim=3,nst=6
  INTEGER               :: loaded_nodes,iel,i,iters,limit
  INTEGER               :: nn,nr,nip,nod,nels,ndof,npes_pp
  INTEGER               :: node_end,node_start,nodes_pp
  INTEGER               :: meshgen
  REAL(iwp)             :: e,v,det,tol,up,alpha,beta,tload
  REAL(iwp),PARAMETER   :: zero = 0.0_iwp
  CHARACTER(LEN=15)     :: element
  CHARACTER(LEN=50)     :: program_name='p121'
  CHARACTER(LEN=50)     :: fname,job_name='p121',label
  LOGICAL               :: converged = .false.

  !Additional variables
  INTEGER               :: fi,fk,fj,c,k,j,l,nlface,mc,mc2,iproc,ierr
  INTEGER               :: snod,urdof,size_nftot,statu(MPI_STATUS_SIZE)
  INTEGER               :: leltvar,na_elt
  INTEGER               :: dummy,dummy_r,bufsize1,bufsize1_r,nels_pp_r
  INTEGER               :: tpos,nrot,inod,inels=0,patchnode,pnc
  REAL(iwp)             :: xinorm,zenorm,P(4,1),Pt(1,4),A(4,4),b(4,6),&
                           Atot(4,4),btot(4,6),a(4,6)

!-----------------------------------------------------------------------------
! 2. Declare dynamic arrays
!-----------------------------------------------------------------------------

  INTEGER,  ALLOCATABLE :: rest(:,:),g_num_pp(:,:),g_g_pp(:,:),node(:)
  REAL(iwp),ALLOCATABLE :: points(:,:),dee(:,:),weights(:),val(:,:),disp_pp(:)
  REAL(iwp),ALLOCATABLE :: g_coord_pp(:,:,:),jac(:,:),der(:,:),deriv(:,:)
  REAL(iwp),ALLOCATABLE :: bee(:,:),storkm_pp(:,:,:),eld(:),eps(:),sigma(:)
  REAL(iwp),ALLOCATABLE :: timest(:)
  REAL(iwp),ALLOCATABLE :: diag_precon_tmp(:,:),eld_pp(:,:),tensor_pp(:,:,:)

  !Additional dynamic arrays
  INTEGER, ALLOCATABLE  :: bc(:,:),g_num(:,:),eurdof(:),eltptr(:),eltvar(:)
  INTEGER, ALLOCATABLE  :: redtpl(:),gtpl(:),gtpl_r(:),redtpl_pp(:),gtpl_pp(:)
  INTEGER, ALLOCATABLE  :: pstore(:,:),pestore(:,:),perstore(:,:)
  REAL(iwp),ALLOCATABLE :: bee_t(:,:),fun(:),fun_mat(:,:),g_coord_pp_v(:)
  REAL(iwp),ALLOCATABLE :: stress_pp(:),g_coord_pp_ip(:,:,:),eig_v(:,:,:,:),ndis(:,:)
```

```
  REAL(iwp),ALLOCATABLE :: redk_pp(:,:,:),redf(:),f(:),redk_pp_r(:,:,:),varea(:),&
                           bftot(:)
  REAL(iwp),ALLOCATABLE :: sigma_mat(:,:,:,:),tensor(:,:,:),d(:,:,:),&
                           tensor_r(:,:,:),g_coord_pp_ip_r(:,:,:)


!-----------------------------------------------------------------------
! 3. Read job_name from the command line.
!    Read control data, mesh data, boundary and loading conditions.
!-----------------------------------------------------------------------

  ALLOCATE(timest(20))
  timest    = zero
  timest(1) = elap_time()

  CALL find_pe_procs(numpe,npes)!MPI Initialised in this function

  CALL read_p121(job_name,numpe,e,element,limit,loaded_nodes,meshgen,nels,nip,&
                 nn,nod,nr,tol,v,nlface)

  CALL calc_nels_pp(nels,iel_start)

  ndof = nod*nodof
  ntot = ndof

  IF(nod==8)THEN
    snod=4
  ELSEIF(nod==20)THEN
    snod=8
  END IF

  ALLOCATE(rest(nr,nodof+1),g_num_pp(nod,nels_pp),g_coord_pp(nod,ndim,nels_pp),&
           bc(nn,nodof+1),g_num(nod,nels),eurdof(nels),eltptr(nels+1))

  g_num_pp   = 0;     g_coord_pp = 0;     rest       = 0
  eurdof     = 0;     leltvar    = 0;     na_elt     = 0
  urdof      = 0;     k          = 1;     bc         = 0

  CALL read_g_num_pp(job_name,iel_start,nels,nn,numpe,g_num_pp,g_num)

  IF(meshgen == 2) CALL abaqus2sg(element,g_num_pp)

  CALL read_g_coord_pp(job_name,g_num_pp,nn,npes,numpe,g_coord_pp)
```

```
  CALL read_rest(job_name,numpe,rest)

!------------------------------------------------------------------------
! 4.  Initialise MUMPS
!------------------------------------------------------------------------

  id%COMM = MPI_COMM_WORLD
  id%SYM =  2 !Unsymmetric matrix
  id%PAR =  1 !Host is involed in factorisation phase
  id%JOB = -1 !Call to MUMPS initialisation phase

  CALL DMUMPS(id) !Requires MPI to be initiailised

  CALL calc_urdof(nn,bc,rest,urdof,nels,nod,g_num,eurdof,leltvar,na_elt,eltptr)

  IF(numpe==1)THEN
    id%N=urdof
  END IF

  ALLOCATE(eltvar(leltvar))

!------------------------------------------------------------------------
! 5. Allocate dynamic arrays used in main program
!------------------------------------------------------------------------

  ALLOCATE(points(nip,ndim),dee(nst,nst),jac(ndim,ndim),                    &
           der(ndim,nod),deriv(ndim,nod),eld_pp(ntot,nels_pp),bee(nst,ntot), &
           storkm_pp(ntot,ntot,nels_pp),eld(ntot),eps(nst),sigma(nst),       &
           weights(nip),g_g_pp(ntot,nels_pp),fun(nod),fun_mat(ndim,ndof),    &
           redf(urdof), g_coord_pp_v(ndof),g_coord_pp_ip(ndim,nip,nels_pp),  &
           bee_t(ntot,6), redtpl(urdof),gtpl(leltvar),gtpl_r(leltvar),       &
           redk_pp(ntot,ntot,nels_pp),redk_pp_r(ntot,ntot,nels_pp),          &
           ndis(nn,ndim),varea(nodof))

  fun          = 0;        fun_mat       = 0;
  g_coord_pp_v = 0;        g_coord_pp_ip = 0

!------------------------------------------------------------------------
! 6. Loop the elements to find the steering array and the number of
!    equations to solve.
!------------------------------------------------------------------------
```

```
  CALL rearrange(rest)

  g_g_pp = 0

  elements_1: DO iel = 1, nels_pp
    CALL find_g(g_num_pp(:,iel),g_g_pp(:,iel),rest)
  END DO elements_1

  neq = 0

  elements_2: DO iel = 1, nels_pp
    i = MAXVAL(g_g_pp(:,iel))
    IF(i > neq) neq = i
  END DO elements_2

  neq = MAX_INTEGER_P(neq)

!------------------------------------------------------------------------
! 7. Create interprocessor communication tables
!------------------------------------------------------------------------
  CALL calc_neq_pp
  CALL calc_npes_pp(npes,npes_pp)
  CALL make_ggl(npes_pp,npes,g_g_pp)

!------------------------------------------------------------------------
! 8. Element stiffness integration and storage
!------------------------------------------------------------------------

  CALL deemat(e,v,dee)
  CALL sample(element,points,weights)

  storkm_pp       = zero
  diag_precon_tmp = zero
  dummy           = 1
  gtpl            = 0
  redk_pp         = 0

  elements_3: DO iel=1,nels_pp
    gauss_pts_1: DO i=1,nip
      CALL shape_der(der,points,i)
      jac   = MATMUL(der,g_coord_pp(:,:,iel))
      det   = determinant(jac)
      CALL invert(jac)
```

```
      deriv = MATMUL(jac,der)
      CALL beemat(deriv,bee,bee_t)
      storkm_pp(:,:,iel) = storkm_pp(:,:,iel) +                          &
                           MATMUL(MATMUL(TRANSPOSE(bee),dee),bee) *    &
                           det*weights(i)
    END DO gauss_pts_1

    CALL calc_ke_red(nn,nod,g_num_pp,iel,bc,redk_pp,storkm_pp,redtpl,&
                     g_g_pp,dummy,gtpl)

  END DO elements_3

  CALL calc_eltvar(dummy,eltvar,npes,gtpl,gtpl_r)

  CALL calc_NZ(id%NZ,id%NZ_loc,nels_pp,iel_start,nels,eltptr)

!------------------------------------------------------------------------
! 9. Get force vector
!------------------------------------------------------------------------


  IF(loaded_nodes > 0 .OR.nlface>0) THEN

    CALL read_loads2(job_name,numpe,nlface,loaded_nodes,snod,nod,nn,size_nftot)

    ALLOCATE(node(size_nftot),val(3,size_nftot),f(nn*nodof),bftot(nn*3))

    val    = 0
    node   = 0
    f      = 0
    c      = 0

    CALL read_loads(job_name,numpe,node,val,loaded_nodes,nlface,snod,nod,nn,size_nftot)

    CALL bodyforce(element,points,weights,nels,nn,nod,bftot)

    DO i=1,size_nftot
      DO j=1,3
        k=node(i)
        f(((k-1)*3)+j)=val(j,i)
      END DO
    END DO
```

```
    DO i=1,nn
      DO j=1,3
        IF(bc(i,j+1)>0)THEN
          c=c+1
          redf(c)=f(((i-1)*3)+j)
        END IF
      END DO
    END DO
    tload = SUM(redf)

    DEALLOCATE(node,val)

  END IF

!---------------------------------------------------------------------------
! 9. MUMPS SOLVER
!---------------------------------------------------------------------------

  IF(numpe==1)THEN
    ALLOCATE(id%IRN(id%NZ),id%JCN(id%NZ))
  END IF
  j=0
  mc=0
  k=0

  IF(numpe==1)THEN
    DO iel=1,nels
      k=0
      DO i=eltptr(iel),eltptr(iel+1)-1
      k=k+1
      !DO j=eltptr(iel),eltptr(iel+1)-1
        DO j=eltptr(iel),eltptr(iel)+k-1
          mc=mc+1
          id%IRN(mc)=eltvar(i)
          id%JCN(mc)=eltvar(j)
        END DO
      END DO
    END DO
  END IF

  id%ICNTL(18)=2
  id%ICNTL(7)=5
  id%JOB =1
```

```
CALL DMUMPS(id)

IF(numpe==1)THEN
  DEALLOCATE(id%IRN,id%JCN)
END IF

ALLOCATE(id%IRN_loc(id%NZ_loc),id%JCN_loc(id%NZ_loc),id%A_loc(id%NZ_loc))

mc=0
DO iel=1,nels_pp
   DO i=1,eltptr(iel_start+iel)-eltptr(iel_start+iel-1)
       DO j=1,i
         mc = mc+1
         id%A_loc(mc)=redk_pp(i,j,iel)
      END DO
   END DO
END DO

mc = 0

DO iel=1,nels_pp
k=0
  DO i=eltptr(iel_start+iel-1),eltptr(iel_start+iel)-1
  k=k+1
      DO j=eltptr(iel_start+iel-1),eltptr(iel_start+iel-1)+k-1
      mc=mc+1
      id%IRN_loc(mc)=eltvar(i)
      id%JCN_loc(mc)=eltvar(j)
    END DO
  END DO
END DO

id%JOB =2
CALL DMUMPS(id)

IF(numpe==1)THEN
  ALLOCATE(id%RHS(id%N))
  id%RHS=redf
  DEALLOCATE(redf)
END IF

id%JOB      =3
CALL DMUMPS(id)
```

```
  DEALLOCATE(id%A_loc,id%JCN_loc,id%IRN_loc,eltvar)

  IF(numpe==1)THEN

    k = 0

    DO i=1,nn
      DO j=1,3
        IF(bc(i,j+1).GT.0)THEN
          k=k+1
          ndis(i,j)=id%RHS(k)
        END IF
      END DO
    END DO
    DEALLOCATE(id%RHS)

    !Nodal displacement output written to file
    OPEN(7, FILE='mumps_dis_output', STATUS='replace', ACTION='write')

    WRITE(7,*)'*DISPLACEMENT, CALCULATED USING MUMPS LINEAR SOLVER'

    DO i=1,nn
      WRITE(7,'(i8,3(1p,e14.4))')i,ndis(i,:)
    END DO

    DEALLOCATE(ndis)

  END IF

  id%JOB = -2

  CALL DMUMPS(id)

!-------------------------------------------------------------------------------
! 12. Recover stresses at centroidal gauss point
!-------------------------------------------------------------------------------

  ALLOCATE(tensor_pp(nst,nip,nels_pp))

  tensor_pp = zero
  nip       = 8
  eld_pp    = zero
```

```
  CALL gather(xnew_pp,eld_pp)

  Calculates displacement per processor required to calculated stresses at gps
  DO i=1,nels_pp
    DO j=1,nod
      k=g_num_pp(j,i)
        DO l=1,3
        eld_pp((3*(j-1))+l,i)=ndis(k,l)
      END DO
    END DO
  END DO
  elements_6: DO iel=1,nels_pp
    gauss_pts_2: DO i=1,nip
      CALL shape_der(der,points,i)
      jac   = MATMUL(der,g_coord_pp(:,:,iel))
      CALL invert(jac)
      deriv = MATMUL(jac,der)
      CALL beemat(deriv,bee,bee_t)
      eps   = MATMUL(bee,eld_pp(:,1))
      sigma = MATMUL(dee,eps)
      tensor_pp(:,i,iel)=sigma
     END DO gauss_pts_2
  END DO elements_6

!------------------------------------------------------------------------------
! Gauss Point Information - Added by Lorna
!------------------------------------------------------------------------------

  DO iel=1,nels_pp
    DO i=1,nip
    CALL shape_fun(fun,points,i)
    fk=0
      DO fi=1,ndof,3
        fk=fk+1
        DO fj=1,3
          fun_mat(fj,fi+fj-1)   = fun(fk)
          g_coord_pp_v(fi+fj-1) = g_coord_pp(fk,fj,iel)
        END DO
      END DO
      g_coord_pp_ip(:,i,iel) = MATMUL(fun_mat,g_coord_pp_v)
    END DO
  END DO
```

```
  IF (numpe==1)THEN
    fname = job_name(1:INDEX(job_name," ")-1)//".gp"
    OPEN(25,file=fname,status='replace',action='write')
  END IF

  label = "*GAUSS POINT INFORMATION"
  CALL WRITE_GAUSS_INFO(label,25,1,nip,nels_pp,npes,numpe,ndim,g_coord_pp_ip,tensor_pp)

!------------------------------------------------------------------------------
! SPC Error Analysis - Added by Lorna
!------------------------------------------------------------------------------

 nip = 1
 deallocate(points,weights,tensor_pp,g_coord_pp_ip)
 allocate(points(nip,ndim),weights(nip),tensor_pp(nst,nip,nels_pp),&
          sigma_mat(3,3,nip,nels),tensor(nst,nip,nels),&
          d(ndim,nip,nels),eig_v(ndim,ndim,nip,nels),&
          g_coord_pp_ip(ndim,nip,nels_pp),g_coord_ip(ndim,nip,nels)&
          tensor_pp_r(nst,nip,nels_pp),g_coord_pp_ip_r(ndim,nip,nels_pp))
 call sample(element, points, weights)

  DO iel=1,nels_pp
    DO i=1,nip
      CALL shape_der(der,points,i)
      jac   = MATMUL(der,g_coord_pp(:,:,iel))
      CALL invert(jac)
      deriv = MATMUL(jac,der)
      CALL beemat(deriv,bee,bee_t)
      eps   = MATMUL(bee,eld_pp(:,1))
      sigma = MATMUL(dee,eps)
      tensor_pp(:,i,iel)=sigma
     END DO gauss_pts_2
  END DO elements_6

  DO iel=1,nels_pp
    DO i=1,nip
    CALL shape_fun(fun,points,i)
    fk=0
      DO fi=1,ndof,3
        fk=fk+1
        DO fj=1,3
          fun_mat(fj,fi+fj-1)   = fun(fk)
```

```
          g_coord_pp_v(fi+fj-1) = g_coord_pp(fk,fj,iel)
       END DO
      END DO
      g_coord_pp_ip(:,i,iel) = MATMUL(fun_mat,g_coord_pp_v)
    END DO
  END DO

IF (numpe==1)THEN
  tensor(:,:,1:nels_pp)=tensor_pp(:,:,:)
  g_coord_ip(:,:,1:nels_pp)=g_coord_pp_ip(:,:,:)
  tpos=nels_pp
END IF

DO iproc = 2,npes
  IF (numpe==iproc) THEN
    bufsize1 = nip*2*ndim*nels_pp
    bufsize2 = nip*ndim*nels_pp
    CALL MPI_SEND(bufsize1,1,MPI_INTEGER,0,iproc+npes,MPI_COMM_WORLD,ier)
    CALL MPI_SEND(bufsize2,1,MPI_INTEGER,0,iproc+npes,MPI_COMM_WORLD,ier)
    CALL MPI_SEND(tensor_pp,bufsize1,MPI_REAL8,0,iproc+npes,MPI_COMM_WORLD,ier)
    CALL MPI_SEND(g_coord_ip_pp,bufsize2,MPI_REAL8,0,iproc,MPI_COMM_WORLD,ier)
  END IF
  IF (numpe==1)THEN
    CALL MPI_RECV(bufsize1,1,MPI_INTEGER,iproc-1,iproc+npes,MPI_COMM_WORLD,statu,ier)
    CALL MPI_RECV(bufsize2,1,MPI_INTEGER,iproc-1,iproc+npes,MPI_COMM_WORLD,statu,ier)
    CALL MPI_RECV(tensor_pp_r,bufsize1,MPI_REAL8,iproc-1,iproc+npes,MPI_COMM_WORLD,statu,ier)
    CALL MPI_RECV(g_coord_ip_pp_r,bufsize2,MPI_REAL8,iproc-1,iproc+npes,MPI_COMM_WORLD,statu,ier)
    tensor(:,:,tpos+1:tpos+(bufsize1/(nip*ndim*2)))=tensor_pp_r(:,:,:)
    g_coord_ip(:,:,tpos+1:tpos+(bufsize1/(nip*ndim*2)))=g_coord_pp_ip_r(:,:,:)
    tpos=tpos+(bufsize1/(nip*ndim*2))
  END IF
END DO

DO iel=1,nels
  DO i=1,nip
    sigma_mat(1,1,i,iel)=tensor(1,i,iel);
    sigma_mat(2,2,i,iel)=tensor(2,i,iel);sigma_mat(3,3,i,iel)=tensor(3,i,iel)
    sigma_mat(1,2,i,iel)=tensor(4,i,iel);sigma_mat(2,1,i,iel)=tensor(4,i,iel)
    sigma_mat(2,3,i,iel)=tensor(5,i,iel);sigma_mat(3,2,i,iel)=tensor(5,i,iel)
    sigma_mat(1,3,i,iel)=tensor(6,i,iel);sigma_mat(3,1,i,iel)=tensor(6,i,iel)
    CALL jacobi(sigma_mat(:,:,i,iel),ndim,ndim,d(:,i,iel),eig_v(:,:,i,iel),nrot)
  END DO
END DO
```

```
CALL calc_nodes_pp(nn,npes,numpe,node_end,node_start,nodes_pp)
ALLOCATE(pestore(nodes_pp,9))

DO inod=node_start,node_end
  DO iel=1,nels
    DO k=1,8
      IF(g_num(iel,k)==inod)then
        inels=inels+1
        pestore(inod-node_start+1,inels)=j
      END IF
    END DO
  END DO
  IF(iel==nels)THEN
    pestore(inod-node_start+1,9)=inels
  END IF
END DO

pnc=0
DO inod=node_start,node_end
  IF(pestore(inod-node_start+1,9).gt.0)then
    pnc=pnc+1
  END IF
END DO

allocate(perstore(pnc,10))
pnc=0
DO inod=node_start,node_end
  IF(pstore(inod-node_start+1,9).gt.0)then
   pnc=pnc+1
   perstore(pnc,:)=pestore(inod-node_start+1,:)
   perstore(pnc,10)=inod-node_start+1
  END IF
END DO


DO inod=1,pnc
  btot=0.0
  Atot=0.0
  patchnode=perstore(pnc,10)
  DO iel=1,perstore(inod,9)
    DO i=1,nip
      P(1,1)=1
```

```
       P(2,1)=g_coord_ip(1,i,perstore(inod,iel)
       P(3,1)=g_coord_ip(2,i,perstore(inod,iel)
       P(4,1)=g_coord_ip(3,i,perstore(inod,iel)
       Pt(1,1)=P(1,1);Pt(1,2)=P(2,1);Pt(1,3)=P(3,1);Pt(1,4)=P(4,1)
       A=Pt*P
       Atot=A+Atot
       b=Pt*tensor(1,i,iel)
       btot=b+btot
     END DO
   END DO
 END DO

!------------------------------------------------------------------------------
! 14. End Program
!------------------------------------------------------------------------------

  CALL WRITE_P121m(iters,job_name,neq,nn,npes,nr,numpe,timest,tload)

  CALL shutdown()

END PROGRAM p121
```

References

[1] British energy generation (uk) ltd.

[2] P. Amestoy, I. Duff, J. Koster, and J. Y. L'Excellent. A fully asynchronous multifrontal solver using distributed dynamic scheduling. *SIAM Journal of Matrix Analysis and Applications*, 23:15–41, 2001.

[3] P. Amestoy, A. Guermouche, J. Y. L'Excellent, and S. Pralet. Hybrid scheduling for the parallel solution of linear systems. *Parallel Computing*, 32(2):136–156, 2006.

[4] K. Bathe, J. Walczak, A. Welch, and N. Mistry. Non-linear analysis of concrete structures. *Computers and Structures*, 32:564–590, 1989.

[5] W. Chen. *Plasticity in reinforced concrete*. McGraw-Hill Book Comapny, 1982.

[6] CIS_DUrham_University. The hamilton cluster: technical specification. Online at `www.dur.ac.uk/its/local/hpc/hamilton/technical_spec`, September 2011 [Accessed: 10 January 2011].

[7] M. Ebeida, A. Patney, J. Owens, and E. Mestreau. Isotropic conforming refinement of quadrilateral and hexahedral meshes using two-refinement templates. *International Journal For Numerical Methods In Engineering*, 2011.

[8] C. Geuxaine and J.-F. Remacle. Gmsh 2.5 reference manual. Online at `http://geuz.org/gmsh/`, October 2010 [Accessed: 12 March 2011].

[9] N. Harris, S. Benzley, and S. Owen. Conformal refinement of all-hexahdral element meshes based on multiple twist plan insertion. 13*th International Meshing Roundtable*, pages 157–168, 2004.

[10] E. Hinton and D. Owen. *Finite element programming*. Academic Press, 1977.

[11] Y. Ito, A. Shih, and B. Soni. Octree-based reasonable-quality hexahedral mesh generation using a new set of refinement templates. *International Journal For Numerical Methods In Engineering*, 77:1809–1833, 2009.

[12] H. Kupfer. Das verhalten des betons unter mehrachsigen kurzzeitbelastung unter besonderer berticksichigung der zweiachsigen beanspruchung, *desch. Ausschuss Stahlbeton*, 229, 1973.

[13] H. Kupfer, H. Hilsdorf, and Rüsch. Behaviour of concree under biaxial stresses. *Journal of the American Concree Institute*, 66(8):656–666, 1969.

[14] T. Li and R. Crouch. A c2 plasticity model for structural concrete. *Computers and Structures*, 88:1322–1332, 2010.

[15] L. Marechal. New approach to octree-based hexahedral meshing. 10*th International Meshing Roundtable*, pages 209–221, 2001.

[16] L. Margetts. *Programming the finite element method*, PhD thesis, University of Manchester, UK, 2002.

[17] S. Mitchell and T. Tautges. Pillowing doublets: refining a mesh to ensure that faces share at most one edge. 4*th International Meshing Roundtable*, pages 231–240, 1996.

[18] MUMPS. Multifrontal massively parallel solver users' guide. Online at `graal.ens-lyon.fr/MUMPS`, February 2010 [Accessed 2 January 2011].

[19] Oracle. Oracle grid engine. Online at
`http://www.oracle.com/us/products/tools/oracle-grid-engine-075549.html`, February 2012 [Accessed: 12 February 2012].

[20] ParaFEM. Parafem: the freely available portable library for parallel finite element analysis. Online at `www.parafem.org.uk`, Septerm 2011 [Accessed: 24 October 2010].

[21] R. Schneiders. Refining quadrilateral and hexahedral element meshes. *Proc. 5th International Conference of Numerical Grid Generation in Computational Field Simulations*, pages 699–708, 1996.

[22] I. Smith and D. Griffiths. *Programming the finite element method*. John Wiley & Sons Ltd, 2004.

[23] K. Tchon and R. C. J. Dompierre. Automated refinement of conformal quadrilateral and hexahedral meshes. *International Journal For Numerical Methods In Engineering*, 59:1539–1562, 2004.

[24] S. Yamakawa and K. Shimada. Hexhoop: Modular templates for converting a hex-dominant mesh to an all-hex mesh. *Engineering with Computers*, 18:211–228, 2002.

[25] H. Zhang and G. Zhao. Adptive hexahedral mesh generation based on local domain curvature and thickness using a modified grid-based method. *Finite Elements in Analysis and Design*, 43:691–704, 2007.

[26] O. Zienkiewicz, R. Taylor, and J. Zhu. *The finite element method its basis & fundamentals*. Elsevier, 2005.

[27] O. Zienkiewicz and J. Zhu. The superconvergent patch recovery and a posteriori error estimates. part 2: Error estimates and adaptivity. *International Journal For Numerical Methods in Engineering*, 33:1365–1382, 1992.

[28] O. Zienkiewicz and J. Zhu. The superconvergent patch recovery and a posteriori error estimator. part 1: The recovery technique. *International Journal For Numerical Methods in Engineering*, 33:1331–1364, 1992.